T · · Systems · · ·

# rvs® portable

Version 5.07

Referenzhandbuch

Die in diesem Handbuch aufgeführten Produkte sind urheberrechtlich geschützt und stehen dem jeweiligen Rechtsinhaber zu.

rvs®

Version 5.07

Referenzhandbuch



Holzhauser Straße 4 – 8

13509 Berlin



Inhaltliche Änderungen dieses Handbuches behalten wir uns ohne Ankündigung vor. T-Systems haftet nicht für technische oder drucktechnische Fehler oder Mängel in diesem Handbuch. Außerdem übernimmt T-Systems keine Haftung für Schäden, die direkt oder indirekt auf Lieferung, Leistung und Nutzung dieses Materials zurückzuführen sind.

# Inhaltsverzeichnis

# Änderunghistorie

Folgende Änderungen wurden im Referenzhandbuch in den einzelnen Versionsschritten durchgeführt:

Version 5.07

- Neuer Parameter im Kapitel 8.1 “Die rvs® Parameter im Überblick”: `RVSDIAEXTENDEDMODE`
- Neuer Wert "D" für Parameter `VFTYP`
- Neue Parameter für `rvssce` (**-f** und **-i**)

Version 5.06

- Neuer Parameter im Kapitel 8.1 “Die rvs® Parameter im Überblick”: `CHECKMAXPSESSIONS`
- Dateiverschlüsselung und Komprimierung mit ComSecure jetzt auch für Dateien > 2 GB möglich
- rvsjs kann jetzt auch auf Windows-Systemen mit der Option `-t` gestartet werden, die festlegt, in welchen Zeitintervallen das Jobverzeichnis auf Jobdateien überprüft werden soll.
  Auf UNIX-Systemen besteht die Möglichkeit, die Konfiguration von rvsjs über die Kommandozeile durchzuführen.
- Kleine Verbesserungen

Version 5.05

- Neue Parameter im Kapitel 8.1 “Die rvs® Parameter im Überblick”: `AUTODECRYP, CALLINGNUMCHECK, CNTIE, DTCOPY, EFIDGAPTIMEOUT, HEAVYDUTY, IECLEANTIME, STATCHECKINT.`

Version 5.04

- Kleine Verbesserungen

Version 5.03

- Kapitel 7.2 CISCO-Konfiguration überarbeitet.

Version 5.0

- Neue Parameter DIALCOUNT und DIALRTIME im Kapitel 8.1, die für die ISDN-Verbindung bei alternativen Netzwerken von Bedeutung sind.
- Neues Kapitel 7 "XOT-Kommunikation", in dem die CISCO-Router-Konfiguration beschrieben ist.
- Neue globale Parameter für Backup/Recovery: REDOMAXSIZE und NUMREDOLOGS (siehe Kapitel über Parameter 8.1)
- Neuer globale Parameter TRACEOFF (siehe Kapitel über Parameter 8.1 und Kapitel 0)
- Neue Parameter zur Steuerung des gleichzeitigen Starts der rvs® Middleware und rvs®: MWSTART, MWSTOP und MWTIMEOUT (siehe Kapitel über Parameter 8.1)
- Neue Parameter DBLOGMAXENTRIES und DBLOGMINENTRIES für die Sicherung der Log-Meldungen auf die Platte (nur im Zusammenhang mit einer externen Datenbank verfügbar, Kapitel über Parameter 8.1).

Version 4.06

- Neue Parameter COMPFLAGS und CRYPFLAGS im Kapitel 8.

Version 4.05:

- Kapitel 8.3 "Robustheit von rvs® bezüglich Speicherplatz".

Version 4.02:

- Kapitel 8 "Die rvs® Parameter" aus dem Benutzerhandbuch übernommen, da die Parameter für alle rvs® portable (rvsX, rvsXP und rvs400) identisch sind.
- Kapitel 9.1 "rvsjs" aus dem separaten Dokument übernommen.

Version 4.00:

- Kapitel 0 "Leitungstraces im Speicher"
- Ersetzungsvariablen ?CNIE? und ?CNIZ? im RE (Kapitel 10.7)

Version 3.05:

- Kapitel 2.5 "rvs® Service Provider (rvsSP)",
- Kapitel 9.12 "rvscheckdb"
- Kapitel 2.11 "rvs® Data Center".

# I. Einführung

Das Produkt rvs® gibt es auf vielen verschiedenen Plattformen. Die Produkte rvsX, rvsNT, rvsXP und rvs400 bilden zusammen die Produktgruppe rvs® portable.

Für den täglichen Umgang mit rvs® dienen die rvs® Benutzerhandbücher. Aus Gründen der Benutzerfreundlichkeit wurden unterschiedliche plattformspezifische rvs® Benutzerhandbücher erstellt.

Die Gemeinsamkeiten aller rvs® Varianten aus der Produktgruppe rvs® portable werden in diesem Handbuch beschrieben. Außerdem wird noch detaillierter als im Benutzerhandbuch auf die technische Grundlagen von rvs® eingegangen.

Dieses Kapitel beinhaltet eine kurze Beschreibung des rvs® Systems, sowie eine Erklärung der Darstellungskonventionen, die in diesem Handbuch benutzt werden.

# 1 rvs® und seine Schnittstellen im Überblick

rvs® bietet einen zuverlässigen Datenübertragungsdienst, in dem es als eigenständiges Werkzeug Daten senden, empfangen und verteilen kann. rvs® kann in Anwendungen für die Automatisierung des Datenaustausches zwischen Netzknoten und Benutzern integriert werden. Typische Anwendungsbereiche von rvs® sind EDI (**E**lectronic **D**ocument **I**nterchange), CAD (**C**omputer **A**ided **D**esign), Finanztransaktionssysteme und sichere Übertragung von Stammdaten und Daten von Medienkonzernen.

Für verschiedene Grade der Automatisierung sind passende Schnittstellen zu rvs® für die Erzeugung von Sende- und Empfangseinträgen verfügbar:

| | |
|---|---|
| **Dialog Schnittstelle** `rvsdia` | Ist ein interaktives Werkzeug für die Erzeugung jeweils einzelner Einträge; Abfragefunktionen informieren Sie über den Status Ihrer Anforderungen. |
| **Kommandozeilen-Schnittstelle** `rvsbat` | Liest Kommandos aus einer Datei. Diese Eingabedatei ist eine einfache Textdatei, die mit einem beliebigen Editor vorbereitet oder als Ausgabedatei eines Anwendungsprogrammes erzeugt werden kann. |
| **C-CAL-Schnittstelle** `rvscal` | Erlaubt Anwendungsprogrammen durch Aufrufe von Funktionen der Programmiersprache C die direkte Generierung von rvs® Kommandoeinträgen. |
| **J-CAL-Schnittstelle** | Diese Java-Schnittstelle wurde im Zusammenhang mit dem rvs Client Server, einer netzwerkfähigen Erweiterung von rvs® portable, entwickelt. |
| **XML-Schnittstelle** | XML-Schnittstelle ermöglicht Export und Import von rvs® bezogenen Daten im XML-Format. |

rvs® analysiert die Sendeeinträge, erzeugt Sendekommandos und startet die Sendeprozesse, die Ihre Dateien an den oder die richtigen Nachbarknoten im OFTP-Netzwerk für die Zustellung oder die Weiterleitung (Routing) übertragen. Der jeweilige Sendeeintrag bleibt im Wartezustand, bis eine positive Bestätigung vom Empfänger (EERP) kommt.

Empfangsprozesse nehmen eintreffende Dateien an und legen sie in temporären Dateien ab. Danach werden die Dateien dem lokalen Empfänger zugestellt oder wenn nötig, werden passende Sendeeinträge für die Weiterleitung an ferne Stationen erzeugt.

Die Zustellung eintreffender Dateien kann durch residente Empfangseinträge beeinflusst werden, die rvs® mitteilen, unter welchem Namen die Datei gespeichert werden soll und ob ein Job für die Weiterverarbeitung der eintreffenden Daten gestartet werden soll.

Jede der oben vorgestellten Schnittstellen erzeugt je einen Eintrag für jede Datei.

## 1.1 Repräsentationsmittel

Dieser Abschnitt enthält die Beschreibung, welche Darstellungskonventionen in diesem Handbuch verwendet werden und welche Bedeutung besonders gekennzeichnete Ausdrücke haben.

### Darstellungskonventionen

| | |
|---|---|
| `courier` | Kommandos, Menübefehle, Dateinamen, Pfadnamen, Programme, Beispiele, Script-Dateien, Optionen, Qualifiers, Datensätze, Felder, Modi, Fensternamen, Dialogboxen und Status |
| **FETT und GROSSBUCHSTABIG** | Parameter, Umgebungsvariablen, Variablen |
| "Hochkommata" | Verweise auf andere Handbücher, Kapitel und Abschnitte, Literatur |
| **fett** | wichtige Begriffe, Betriebssystemnamen, Eigennamen, Schaltflächen (Buttons), Funktionstasten |

### Begriffe

rvsX ist das Synonym für rvs® auf **UNIX** Systemen.

rvsNT ist das Synonym für rvs® auf **Windows NT** Systemen.

rvsXP ist das Synonym für rvs® auf **Windows XP / 2000 / Vista / 7 / WS 2003 / WS 2008** Systemen

rvs400 ist das Synonym für rvs® auf **OS/400** Systemen.

### Verzeichnisse

Weil Benutzerverzeichnisse auf unterschiedlichen Plätzen bei den unterschiedlichen Betriebssystemen zu finden sind, benutzen wir in diesem Handbuch die Variable **$RVSPATH**. Die Standardwerte sind:

- `/home/rvs/` für **AIX**, **Solaris**, **IRIX**, **Linux** und **SCO**
- `/users/rvs/` für **HP-UX**
- `/defpath/rvs/` für **SINIX**
- `C:\Programme\rvs`® für **Windows**

Ersetzen Sie diese Variable durch Ihren richtigen Pfad.

Dateinamen bei **OS/400** Systemen werden immer groß geschrieben.

# II. Technischer Überblick

Dieser Teil des Handbuches liefert einen Überblick über die funktionalen Elemente von rvs®, über die Protokollschichten in rvs®, über die Grundbegriffe des LU 6.2, X.25/ISDN und TCP/IP, verbunden mit rvs®.

## 2 Systemkomponenten

### Elemente des portablen rvs®

rvs® besteht aus den folgenden Hauptelementen:

- Monitor (`rvsmon`)
- MasterTransmitter (`rvsxmt`)
- Kommunikationsmodule (`rvscom`)
- LogWriter (`rvslgw`)
- Service Provider (`rvssp`)
- ActivePanel (`rvsap`)
- Operator-Konsole (`rvscns`)
- Dialog-Schnittstelle (`rvsdia`)
- Kommandozeilen-Schnittstelle (`rvsbat`)
- C-Cal-Schnittstelle (`rvscal`), J-CAL-Schnittstelle, XML-Schnittstelle
- Datenbank

### Informationsfluß

In einer laufenden rvs® Umgebung ist die rvs® Datenbank das zentrale Element, in der alle benötigten statischen und dynamischen Informationen abgelegt sind. Statische Informationen sind z.B. die rvs® Parameter, sowie die Stationstabelle, auf welche von der lokalen Station zugegriffen werden kann. Dynamische Informationen sind Informationen über aktuelle Prozesse, wie z.B. Sendeaufträge.

Die Prozesse kommunizieren nicht direkt miteinander, sie nutzten die rvs® Datenbank als Kommunikationsmedium. (Ausnahme: die Operator-Konsole in einigen rvs® Anwendungen kommuniziert direkt mit dem rvs® Monitor). Eine enge Verbindung im Sinne eines gemeinsamen Speichers und Aufgabenunterteilung – wie bei rvsMVS – ist aus Gründen der Mobilität und der Wartung nicht möglich. Das Ziel war, bei den unterstützten Plattformen so viel wie möglich identischen Code zu nutzen. Nur wenige Systeme unterstützen einen gemeinsamen Speicher und Aufgabenunterteilung, dann jedoch könnte der Code nicht portabel gehalten werden.

Mehrprozeßfähigkeit und externe Datenhaltung sind üblich für alle Systeme, mit Ausnahme von PCs mit MS-DOS oder PC-DOS.

Um den stetig steigenden Datenverkehr effizient zu bewältigen und die Performance von rvs® zu erhöhen, kann die rvs-interne C-ISAM-Datenbank durch eine hoch performante Datenbank ersetzt werden.

- Auf Windows-, AIX- und LINUX-Systemen ab Version 2.05 gibt es die Möglichkeit der Anbindung an eine Oracle-Datenbank.
- Windows-Systeme ab Version 2.11 können an einen Microsoft SQL Server angebunden werden.

## 2.1 rvs® Monitor

Dieser Abschnitt beschreibt die Basiseigenschaften des Monitors, das Ausführen von Sendeaufträgen und das Bearbeiten von Empfangsdaten.

### 2.1.1 rvs® Monitor - Basiseigenschaften

Der rvs® Monitor ist das Modul, das alle anstehenden Aufgaben ausführt und das auf externe Ereignisse reagiert. Der rvs® Monitor ist die zentrale Komponente von rvs®. Seine Hauptaufgaben sind:

- Ausführen von Aufgaben,
- Ausführen von Operator-Kommandos
- Durchsuchen der rvs® Datenbank nach Sendeaufträgen, die auszuführen sind,
- Bereitstellen eines MasterTransmitter-Prozesses, wenn kein solcher aktiv ist, aber Daten zur Übertragung bereitstehen,
- Wiederanlauf von mißglückten Übertragungen
- Weiterbefördern der empfangenen Informationen an den Endempfänger (Benutzer oder anderer Stationen),
- Aktivierung von Jobs, wenn nach dem Empfang einer Datei ein entsprechender residenter Empfangseintrag gefunden wird,
- Schreiben von Statistiken,
- Generieren von Benutzerbenachrichtigungen
- Modifizieren von Parametern und Stationstabelleneinträgen in der rvs® Datenbank auf Operatoraufforderung,
- Wiederherstellen der rvs® Datenbank.

Der rvs® Monitor wird mit Hilfe einer Operator-Konsole gesteuert. Abhängig vom lokalen System ist die Operator-Konsole mit dem rvs® Monitor durch einen Thread oder eine Pipe eng verbunden oder sie stellt ihre Befehle in die rvs® Datenbank (OS/400 oder UNIX).

Der rvs® Monitor durchsucht periodisch die rvs® Datenbank nach externen Ereignissen oder nach Arbeitsaufgaben, die ausgeführt werden können. Wenn keine Aufträge anstehen, schaltet sich der Monitor für eine benutzerdefinierte Zeitspanne ab (Parameter **SLEEP**). Externe Ereignisse, wie die Eingabe eines Operator-

Kommandos oder die Erzeugung eines neuen Sendeauftrages werden dem rvs® Monitor mittels eines Semaphors mitgeteilt und verursachen sein sofortiges "Aufwachen". Der rvs® Monitor konvertiert den Auftrag in ein rvs® Kommando und weist diesem eine eindeutige Kommando-Nummer zu.

Die Weiterverarbeitung von allen rvs® Kommandos wird vom entsprechenden Kommando-Status sowie der Prioritätsinformation gesteuert. Jedes interne Kommando bekommt eine bestimmte Prioritätsstufe, die vom Administrator verändert werden kann.

Die Hauptaufgabe des rvs® Monitors besteht darin, die entsprechenden rvs® Kommandos und deren Status zu bearbeiten. Die wichtigsten Kommandos oder Ereignisse sind:

| | |
|---|---|
| SE | 'SendeEintrag' |
| SK | 'SendeKommando' |
| QS | 'QuittierungsSendung' |
| QE | 'QuittierungsEingang' |
| IE | 'InformationsEingang' |
| IZ | 'InformationsZustellung' |
| OK | 'Operator-Kommando' |
| EC | 'EndeKommando' |

Die Liste der möglichen Status ist:

- \- Ereignis nicht vorhanden
- q wartend (`queued`), noch nicht vom Monitor bearbeitet
- f fertig zum Senden (`forwardable`), wartet auf Übertragungsprozess (MasterTransmitter)
- a aktiv (`active`), in Bearbeitung vom Monitor
- i im Durchlauf (`in transit`), MastrerTransmitter bearbeitet aktuell den Auftrag
- p wartend (`pending`), wartet auf Beendigung
- e beendet (`ended`)
- d aus der Datenbank gelöscht (`deleted`)
- h vom System oder Benutzer angehalten (`held`)
- s der Verkehr zum Ziel ist unterbrochen (suspended) (`SK`, `QS`)
- c Bearbeitung durch ServiceProvider
- x bei Serialisierung, so lange die Einträge nicht bearbeitet werden
- y „Lücke“ beim EFID (EndFileID), wenn ein Fehler auftritt

Im Allgemeinen durchlaufen alle Kommandos die folgende Kommandokette: `q` oder `f`, `a` oder `i`, (`p`), `e`. Einige Status halten so kurz an, dass sie nur kurz zu sehen sind.

### 2.1.2 Bearbeiten eines Sendeauftrages

Ein Benutzer hat einen Sendeauftrag SE in die rvs® Datenbank eingegeben. Der Sendeauftrag wird bei der nächsten Durchsuchung der rvs® Datenbank gefunden, interpretiert und analysiert. Daraus entsteht ein Sendekommando SK, eine Arbeitsaufgabe, die zur Weiterverarbeitung bereitsteht.

Der Sendeauftrag SE hat jetzt den Status `forwardable` erreicht. Der rvs® Monitor aktiviert den MasterTransmitter, falls der noch nicht aktiv ist. Das Sendekommando SK wird ausgeführt, indem die Sendeaufgabe vom MasterTransmitter erfüllt wird.

Das Sendekommando SK hat jetzt den Status `in transit`, der entsprechende Sendeeintrag den Status `pending`. Der Abschluß der Sendeaufgabe, sei es erfolgreich oder nicht, wird dem rvs® Monitor über die rvs® Datenbank mitgeteilt. Neben einer lesbaren Meldung im Log-Buch aktualisiert der Sender den Status und die Fehlerfelder des SK.

Der rvs® Monitor zeigt die Meldung auf der Operator-Konsole an und bewirkt einen Wiederanlauf, wenn in den Fehlerfelder des SK ein Fehler angezeigt wird. Bei Erfolg hat das SK den Status `ended` erreicht, der SE hat immer noch den Status `pending`. Nach dem eventuellen Empfang der Quittierung des Partners, QE (die Odette End-zu-End für eine erfolgreiche Übertragung), kennzeichnet der rvs® Monitor den ursprünglichen SE als `ended`.

### 2.1.3 Bearbeitung ankommender Daten

Ein Empfangsprogramm wird von der fernen Station gestartet, wenn SNA LU 6.2 Kommunikationstechnik benutzt wird.

In anderen Fällen, z.B. reiner X.25 Kommunikation, wartet eine Anzahl von vorgestarteten Empfangsprogrammen auf ankommende Rufe. Der Abschluß des Empfangsprozesses wird in der Datenbank durch das Generieren eines Ereigniseintrages über die eingehende Information `IE` angezeigt.

Die empfangenen Dateien werden in einem temporären Verzeichnis gespeichert. Wenn der rvs® Monitor ein `IE` Kommando entdeckt hat, startet er die entsprechende Aktion, z.B. die Lieferung der Datei an den Empfänger (es können mehrere sein), indem er ein internes `IZ` Kommando generiert.

Das Ausführen des `IZ` Kommandos besteht in der Regel im Kopieren der empfangenen Daten von der temporären Datei und der anschließenden Überführung des `IZ` Kommandos und des entsprechenden `IE` in den Status `ended`. Wenn das `IE` Informationen enthält, die an eine andere rvs® Station geschickt werden sollen (Routing), wird anstelle von `IZ` ein SE Kommando generiert.

## 2.2 MasterTransmitter

Der MasterTransmitter wurde zur besseren Kontrolle paralleler Übertragungsprozesse eingeführt. Seine Hauptaufgaben sind:

- Steuerung der maximalen Anzahl aktiver Übertragungen (Sender- und Empfängerprozesse), um eine Überlastung des lokalen Systems zu verhindern.
- Warteschlange für anstehende Übertragungen verwalten,
- Zulassen von Übertragungen, falls die maximale Anzahl der zulässigen Übertragungsprozesse noch nicht erreicht ist.
- Kontrolle der maximalen Anzahl der aktiven Sender pro entfernte Station. Diese zukünftige Erweiterung erlaubt es, die Behandlung stationsspezifischer Ressourcegrenzen wie die vorhandene Anzahl von SVCs in einer normalen X.25 Verbindung zu behandeln.
- Ein stationsabhängiger intelligenter Mechanismus, der das Senden von mehr als einer Datei erlaubt, oder sogar das Empfangen von Daten während eines aktiven Übertragungsprozesses.
- Kontrolle von vorgestarteten Empfangsprozessen, z.B. für die X.25 Unterstützung.

## 2.3 Kommunikationsprogramm

Sender- und Empfängerprozesse sind in einem gemeinsamen Kommunikationsprogramm zusammengelegt, das abhängig von den Rufparametern als Sender oder Empfänger funktioniert. Die Rolle als Sender oder Empfänger kann sich dynamisch ändern. Das Kommunikationsprogramm besteht aus vier hierarchisch organisierten Schichten:

- Schicht 1 (oberste Schicht): Kommunikationsmodul (Hauptprogramm)
- Schicht 2: Sender- und Empfängermodule
- Schicht 3: das Odette File Transfer Protokoll (OFTP) Modul
- Schicht 4: das Leitungstreibermodul

Der Kommunikationsprozess wird vom MasterTransmitter als Sender durch die Nummer des entsprechenden Sendekommandos als Input-Parameter aktiviert. Anhand dieser Nummer fragt das Kommunikationsmodul die rvs® Datenbank nach der notwendigen Information ab, z.B. nach der entfernten Station, dem Namen der Datei, sowie nach den LU 6.2; X.257ISDN oder TCP/IP und Odette Parametern, und startet dann die Übertragung. Nach dem Senden fordert es den Partner auf, seine ggf. anstehenden Dateien zu senden.

Nach der Übertragung sucht der Sender nach weiteren Datenpaketen, die zu verschicken sind. Das bedeutet, dass wenn mehrere Datenpakete auf das Versenden zu derselben Station warten, sie die gleiche hergestellte Verbindung nutzen werden.

Der Kommunikationsprozess wird vom MasterTransmitter als Empfänger aktiviert durch einen eingehenden LU 6.2 APPC Fernruf - oder er wurde schon voraktiviert, durch das Warten auf eingehende X.25 oder TCP/IP Rufe.

Die notwendigen Informationen über die zu empfangende Datei, wie den Dateinamen, die Dateigröße sowie den Typ, werden aus den ersten ausgetauschten Kopfdaten ermittelt. Wenn der Empfängerprozess erfolgreich beendet wurde, legt er einen `IE` Eintrag in der rvs® Datenbank ab. Der rvs® Monitor findet ihn und startet das Liefern der Daten, `IZ`, an den Endempfänger.

Der Kommunikationsprozess kann durch das Kommando `activate` aktiviert werden. Er funktioniert dann als Sender und stellt die Verbindung zur gewünschten Station her. Wenn es in der lokalen Station oder in der entfernten Station anstehende Datenpakete gibt, werden sie übertragen, wenn nicht wird die Verbindung beendet.

## 2.4 LogWriter

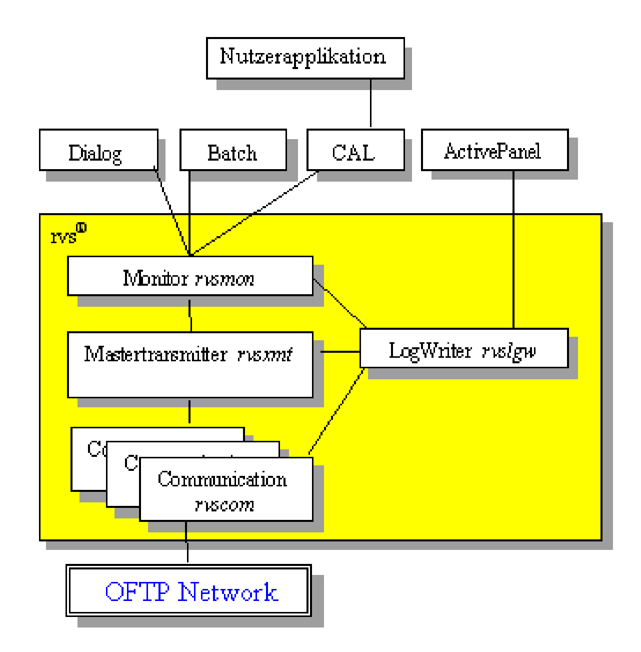
Der LogWriter ist die zentrale Instanz, die Informationen zu Datenübertragungen von anderen rvs® Prozessen empfängt. Der Informationsaustausch zwischen den Prozessen und dem LogWriter erfolgt über Sockets. Daher ist es zwingend erforderlich, dass der TCP/IP Protokollstack aktiv ist (für Windows ggf. über Loopback Adapter).

Diese Datenübertragungs-Informationen werden nach Status- und Ereignisinformationen unterschieden. Statusinformationen sind solche Informationen, die über den Fortgang der gerade aktiven Datenübertragung informieren, also z.B. wieviele Bytes werden in diesem Moment übertragen oder tritt ein Übertragungsfehler auf usw.. Ereignisinformationen sind Informationen, die Auskunft darüber geben, wann z.B. eine Datei übertragen wurde oder welche Dateien übertragen wurden. Diese Art der Informationen bezieht sich also auf Ereignisse, die bereits eingetreten sind.

Abhängig davon, um welche Art es sich handelt, werden die Informationen auf verschiedene Weise angezeigt:

- Die Statusinformationen über eine aktuell stattfindende Datenübertragung können mit einem ActivePanel abgerufen und angezeigt werden.
- Die Ereignisinformationen zwischen einem rvs® Start und Stopp werden in eine temporäre Nachrichtendatei geschrieben und können über die Operator-Konsole abgerufen und angezeigt werden.
- Die Ereignisinformationen unabhängig von einem rvs® Start und Stopp werden in eine dafür definierte Nachrichtendatei gespeichert und können mit Hilfe eines ASCII-Editors angezeigt werden.

Folgende Grafik verdeutlicht die Arbeitsweise des LogWriters:




## 2.5 rvs® Service Provider (rvsSP)

rvs® Service Provider ist ein internes rvs®-Programm (Applikation), welches die folgenden Dienste bereitstellt:

- Komprimierung und Dekomprimierung
- Ver- und Entschlüsselung.

Wenn nötig, werden die Dateien vor dem Senden komprimiert und/oder verschlüsselt und beim Empfang dekomprimiert und/oder entschlüsselt.

`rvsSP` arbeitet eng mit dem rvs®-Monitor zusammen. Dabei behält rvs®-Monitor die Ablaufssteuerung für die Sende-/Empfangsaufträge in rvs® und benutzt `rvsSP` für die Vor- bzw. Nachbearbeitung von Dateien (vor dem Versand bzw. nach dem Empfang).

rvs®-Monitor schreibt alle notwendigen Angaben zu einem Auftrag (Job) an `rvsSP` in jeweils eine Datei pro Auftrag – die Jobdatei. Die Jobdatei wird vom rvs®-Monitor ins Verzeichnis `SPINDIR` gestellt. Danach arbeitet `rvsSP` alle Aufträge aus `SPINDIR` ab. Zwischendateien, die `rvsSP` erzeugt, werden in einem separaten Verzeichnis `SPFILESDIR` abgelegt. Jobs, die fertig bearbeitet sind, legt `rvsSP` ins Verzeichnis

SPOUTDIR ab. rvs®-Monitor wird darüber via IP-Socket-Verbindung informiert und bearbeitet danach alle Jobs, die sich in SPOUTDIR befinden.

**Hinweis**: Die Pfade für die Variablen SPINDIR, SPFILESDIR und SPOUTDIR sind in der Datei $RVSPATH/rvsenv.dat zu setzen. Siehe Benutzerhandbuch, Kapitel 3.11 für die Erklärung von SPINDIR, SPFILESDIR und SPOUTDIR.

## 2.6 ActivePanel

Das ActivePanel ist ein Anzeigeprogramm, mit dem Informationen über aktuell stattfindende Datenübertragungen angezeigt werden können.

Gestartet wird das ActivePanel:

- mit dem Programm `rvsap` unter **UNIX**
- über das Hauptfenster `rvsNT Administrator` Menü `Ansicht` → `Aktive Leitungen` unter **Windows**

## 2.7 Operator-Konsole (`rvscns`)

Die Operator-Konsole benutzen Sie, um die Aktivitäten von rvs® zu steuern. Sie ist die Schnittstelle, die Änderungen in der Datenbank durch die Operator-Kommandos an den rvs® Monitor ermöglicht. Einige der dynamischen Informationen, z.B. Sendekommandos und Sendereihenfolgen können am Bildschirm angezeigt und geändert werden. Andere Kommandos erlauben das Anzeigen und Ändern von rvs® Parametern und der Stationstabelle. Die Änderungen der Stationstabelle werden aus dem eingehenden Datensatz gelesen.

## 2.8 Dialog-Schnittstelle (`rvsdia`)

Die rvs® Dialog-Schnittstelle ist dem wohl bekannten Menüsystem von rvsMVS nachempfunden. Sie ermöglicht das Eintragen, Ändern und Löschen von allen Übertragungen, residenten Empfangseinträgen, Send-Job-Einträgen und Benutzern sowie das Anzeigen der Stati. Die Dialog-Schnittstelle kommuniziert direkt mit der rvs® Datenbank.

Die Dialog-Schnittstelle ist ein endlicher Automat, der für alle portablen rvs® Systeme gleichermaßen benutzt werden kann. Er enthält einen Satz aus Bildschirm-, Eintragungs- und Hilfemasken oder Menüs und eine Datenbankschnittstelle. Die Masken/Menüs sind in allen Systemen, die ANSI-Terminals unterstützen, einsetzbar. Nur für OS/400 werden eigene Menüsysteme benutzt.

## 2.9 Kommandozeilen- und C-Cal-Schnittstelle (`rvsbat` und `rvscal`)

Die Kommandozeilen-Schnittstelle ermöglicht es, die Aufträge als eine einzelne Kommandozeile auf der Ebene der Kommandosprache des entsprechenden Systems einzugeben. Ein solches Kommando kann leicht in die Liste der

Kommandoprozeduren aufgenommen werden. Es kann entweder interaktiv genutzt werden oder in der Kommandoliste eines Datensets, das im Stapelverarbeitungsmodus ausgeführt wird.

Die C-Cal-Schnittstelle. kann direkt mit einem Benutzeranwendungsprogramm verbunden werden. Sie ermöglicht die Eingabe von Aufträgen in die rvs® Datenbank direkt aus dem Benutzeranwendungsprogramm. Ein Beispielprogramm in C und einige Kommandozeilen-Beispiele sind in der vorliegenden rvs® Version integriert.

## 2.10 rvs® Datenbank

Die rvs® Datenbank ist eine relationale Datenbank, die die SQL Sprache unterstützt. Alle rvs® Programme benutzen 'eingebettete' SQL Aufrufe. Die rvs® Datenbank ist aus den folgenden Tabellen aufgebaut:

| | |
|---|---|
| AC | Tabelle der X.28/PAD oder ASCII Parameter |
| BB | Tabelle für Benutzerbenachrichtigungen |
| BT | Tabelle der lokal registrierten rvs® Benutzer (Benutzertabelle) |
| CT | Tabelle der gültigen Konsolen-IDs |
| DB | Tabelle für die aktuelle rvs® und Datenbankversion |
| EC | Tabelle für "EndCommand"-Typ der internen Kommandos |
| ET | Tabelle der gültigen Empfänger (Empfängertabelle) |
| FK | Tabelle der Kommandosfehler |
| FS | Tabelle der Stationenfehler |
| IE | Tabelle der eingegangenen Übertragungen (InformationsEingang) |
| IZ | Tabelle der Zustellungen ("Informationszustellung") |
| JS | Tabelle der Jobstarts nach Sendeversuchen |
| KT | Tabelle der laufenden Kommandos ("Kommandotabelle") |
| LC | Tabelle der letzten eindeutigen Kommandonummer (Letztes Kommando) |
| LD | LastDate (letzte benutzte ODETTE-Zeit) |
| LM | Tabelle der Log-Meldungen |
| LU | Tabelle der LU 6.2 Parameter |
| LT | Tabelle der Log-Meldungen der rvs® Data Center |
| LX | Tabelle für Leitungstreiber Parameter, Erweiterungen. |

| | |
|---|---|
| NK | Tabelle der Nachbarstationen |
| OK | Tabelle der Benutzerkommandos |
| OP | Tabelle der Odette Parameter |
| PT | Tabelle der rvs® Parameter (Parametertabelle) |
| QE | Tabelle der eingegangenen Quittierungen (Quittierungseingang) |
| QS | Tabelle der zu versendenden Quittierungen (QuittierungsSendeeintrag) |
| RE | Tabelle der residenten Empfangseinträge |
| RI | Tabelle der rvs®-Informationen (rvs®-Knoten-Aktivitäten bei rvs® Data Center) |
| RT | Tabelle der routing Informationen (Routingtabelle) |
| SE | Tabelle der Sendeauftagseintragungen (Sendeeintrag) |
| SK | Tabelle der Sendekommandos (Sendekommando) |
| SL | Tabelle der Serialisierungsliste |
| SS | Tabelle der Sendestatistik |
| ST | Tabelle der Stationen ("Stationstabelle") |
| SV | Tabelle der Schlüsselverwaltung |
| TC | Tabelle der TCP/IP Parameter |
| TR | Tabelle der User Level Security |
| VD | Tabelle der zu versendenden Datenpakete (Versanddatei) |
| VM | Tabelle der Sendebenachrichtigungen (Versandmeldung) |
| XP | Tabelle der X.25 Parameter |

Der Hauptschlüssel einer Tabelle ist in der Regel die Stations-ID **SID**. Die Tabellen sind in Zeilen und Spalten organisiert. Die Zeilen enthalten eine Kommando oder Stationsbeschreibung etc., die Spalten enthalten verschiedene Informationseinzelheiten wie Status, Name der Datei, Datum und Uhrzeit des Eintrags etc.

Die physikalische rvs® Datenbank hängt vom lokalen System ab. rvs® unterstützt zur Zeit ISAM, Oracle- und Microsoft SQL-Datenbank. Die physikalischen Unterschiede werden von einer eingebetteten SQL-Schnittstelle verdeckt, die in allen Versionen von rvs® portabel implementiert ist. Das ist für die Mobilität und Wartungsmöglichkeiten von rvs® portabel von entscheidender Bedeutung, da das logische Design nicht von der physikalischen Datei oder Datenbank abhängig ist, die von System zu System erheblich variieren kann.

Die Datenbanktreiber unter der SQL-Schicht sind auch bis zu einem gewissen Grad portabel. Eine ISAM Datenbank und die entsprechenden Treiber werden für Unix, Windows-Systeme und OS/400 ohne extra Kosten bereitgestellt als Teil der Standardlösung.

Um den stetig steigenden Datenverkehr effizient zu bewältigen und die Performanceleistungen von rvs® zu erhöhen, gibt es ab der 2.05 Version auf AIX-, Linux-, Sinix- und Windows NT/XP/2000/Vista/7/WS 2003/WS 2008-Systemen die Möglichkeit der Anbindung an eine Oracle-Datenbank und ab der Version 2.11 auf Windows-Systemen die Möglichkeit der Anbindung an eine MS SQL-Datenbank. Die rvs®-interne C-ISAM-Datenbank wird durch die externe hoch performante Oracle- oder MS SQL Datenbank ersetzt.

Die Datenbankeinträge können mit Hilfe des rvs® Datenbankkommandos `rvsddb` gesichert werden.

## 2.11 rvs® Data Center

In diesem Kapitel wird die technische Basis von rvs® Data Center beschrieben, wohingegen seine Bedienung im Benutzerhandbuch rvsX erläutert wird.

### 2.11.1 Einleitung

rvs® Data Center bietet eine deutlich höhere Ausfallsicherheit und Übertragungskapazität als rvs®.

In einem rvs® Data Center sind mehrere rvs® Server zusammengefasst, um eine hohe Verfügbarkeit (High Availability) des Systems zu gewährleisten.

Abzuarbeitende Aufträge werden gleichmäßig auf alle rvs® Server des rvs® Data Centers verteilt (load balancing).

Die Übertragungskapazität kann erhöht oder gesenkt werden, indem rvs® Server im laufenden Betrieb des rvs® Data Centers hinzugefügt oder entfernt werden (Skalierbarkeit).

Falls einer der Server ausfällt, werden seine Aufgaben von einem anderen Server übernommen, so dass rvs® Data Center störungsfrei weiterfunktionieren kann.

Gegenüber Kommunikationspartnern und bei der Bedienung (z.B. Dateiversand) verhält sich rvs® Data Center wie ein einzelnes rvs®.

### 2.11.2 Systemvoraussetzung

rvs® Data Center steht für rvs®-Version 5.06.00 auf folgenden Plattformen zur Verfügung:

- Window XP / 2000 / 2003 / 7 / Vista / 2008
- AIX 5.3 / 6.1
- Linux i686
- HPUX ia64 / risc
- Sun SPARC Solaris.

Als rvs®-Datenbank wird Oracle oder MS SQL-Server mit folgenden Versionen eingesetzt:

- Windows: MS SQL-Server (alle Versionen), Oracle 10, Oracle 11*)
- AIX 5.3 / 6.1 Oracle 10, Oracle 11
- Linux i686: Oracle 9, Oracle 10, Oracle 11*)
- HPUX ia64: Oracle 9, Oracle 10
- HPUX risc: Oracle 9, Oracle 10*²), Oracle 11*²)
- Sun SPARC Solaris: Oracle 10

*) Oracle 10 Client muss verwendet werden

*²) Oracle 9 Client muss verwendet werden

Auf jedem rvs®-Server (Knoten) muss Oracle-Client-Software installiert sein, um die Zugriffe auf Oracle-Datenbank zu ermöglichen. Normalerweise empfehlen wir die gleiche Versionsnummer für Oracle-Client und Oracle-Server (Ausnahmen s.o.).

Voraussetzung für Zugriffe auf die zentralen Verzeichnisse des rvs® Data Centers im Netzwerk ist das Protokoll NFS (Network File System) Version 3.

### 2.11.3 rvs® Data Center-Architektur

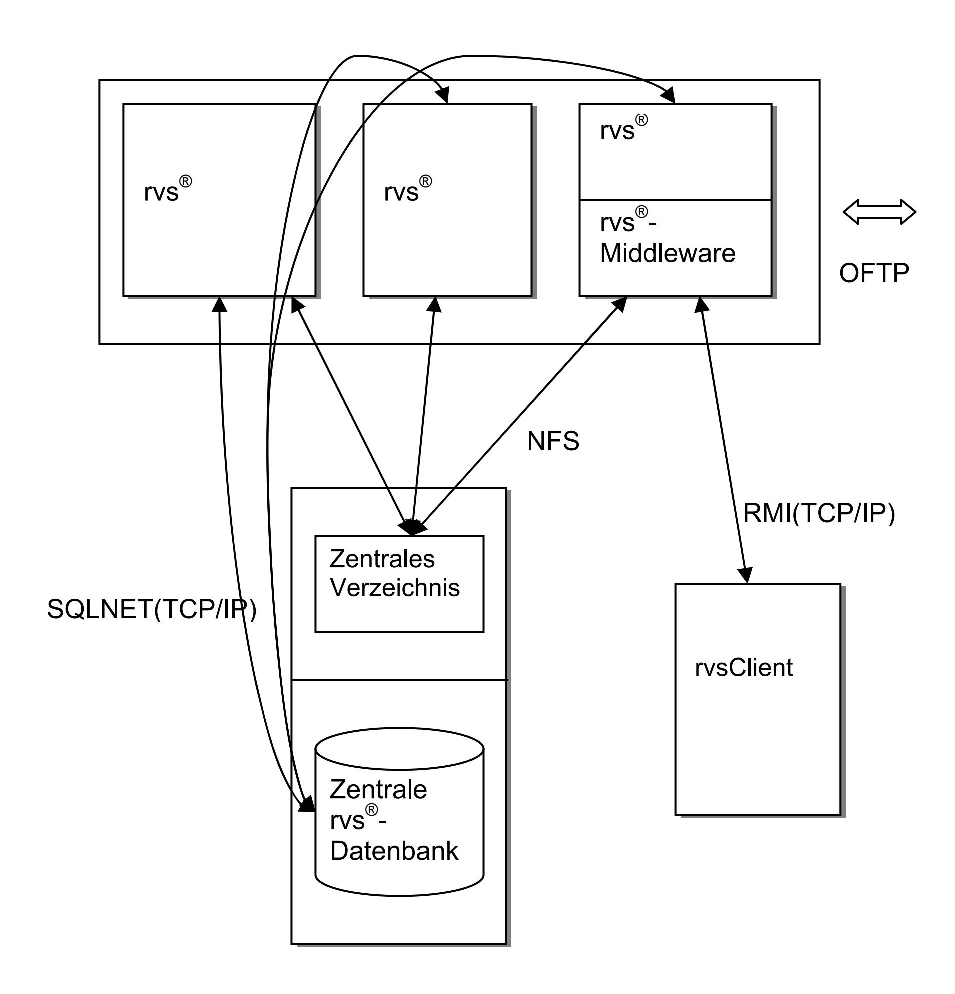
rvs® Data Center besteht aus folgenden Komponenten:

- mehreren rvs®-Knoten (rvs® Servern mit installiertem rvs® und Oracle-Client-Software).
- einer zentralen Oracle-Datenbank, die von den rvs®-Servern per `SQLNET` über TCP/IP erreichbar ist.
- Einem zentralen Verzeichnis, in dem die wichtigsten der von dem rvs® Data Center verwendeten Verzeichnisse enthalten sind. Dieses zentrale Verzeichnis muss über NFS (Network File System) für jeden rvs® Server verfügbar sein. Empfehlenswert ist, dass dieses Verzeichnis auf einem separaten Rechner konfiguriert ist. Es könnte auf dem gleichen Rechner wie die zentrale Datenbank liegen, nicht aber auf einem von den rvs®-Knoten, um die Ausfallsicherheit des gesamten Systems nicht zu gefährden. Folgende Verzeichnisse müssen im zentralen Verzeichnis vorhanden sein:

  - `temp`: für die temporäre Speicherung von Dateien während des Sende- und Empfangsprozesses.
  - `usrdat`: für die Zustellung der empfangenen Dateien.
  - `init`: für die Konfigurationsdateien und Lizenz.
  - `keydir`: für die Schlüssel zur Dateiver- und -Entschlüsselung.
  - `spindir`: für die internen Jobdateien (Komprimierung und Verschlüsselung) des Service Providers, die abgearbeitet werden sollen.
  - `spoutdir`: für die internen Ergebnisdateien (Log-Dateien) der Jobverarbeitung des Service Providers. Die Log-Dateien können zur Fehleranalyse verwendet werden.
  - `spfilesdir`: für die internen Arbeitsdateien des Service Providers, die bei der Jobverarbeitung benutzt werden.
  - Alle zu sendenden Dateien müssen sich auch im zentralen Verzeichnis befinden.
- rvs® Client/Server (welcher seinerseits aus rvs® Middleware und rvs® Client besteht). rvs® Client/Server dient zur Konfiguration und Wartung des rvs® Data Centers. rvs® Middleware muss auf einem beliebigen Knoten laufen, rvs® Client kann hingegen auf einem Rechner außerhalb des rvs® Data Centers ausgeführt werden. Die Kommunikation zwischen Client und Middleware wird durch eine RMI-Verbindung (Java) realisiert. Sollte sich eine Firewall zwischen rvs® Client und rvs® Middleware befinden, müssen sieben frei konfigurierbare Ports freigeschaltet werden.

  Folgende rvs® Client/Server Versionen werden von rvs® unterstützt:
  rvs® Version 5.06 ist mit rvs® Client/Server Version 5.05.00 kompatibel und mit rvs® Client/Server (Client API) 5.05.00 abwärtskompatibel bis zu rvs 3.04 mit rvs Client Server 2.02.

Das folgende Bild soll die rvs® Data Center-Architektur veranschaulichen:




Auf allen rvs®-Knoten müssen Verbindungsnetzwerke (ISDN, X.25 oder TCP/IP) konfiguriert werden, über die Dateien versendet oder empfangen werden. Im obigen Bild ist diese Komponente mit OFTP dargestellt.

### 2.11.4 Benutzerschnittstelle von rvs® Data Center

Dem Benutzer von rvs® Data Center stehen folgende Schnittstellen zur Verfügung:

- rvs® Batch Schnittstelle (`rvsbat`)
- Skripte
- rvs® Client/Server.

`rvsbat` dient hauptsächlich zum automatisierten Versenden von Dateien und zum Anlegen von residenten Empfangseinträgen (REs) und Jobstarts nach Sendeversuchen (JSs). `rvsbat` kann auf einem beliebigen rvs®-Knoten ausgeführt werden. Die Vorgehensweise ist identisch wie bei rvs® standalone, wobei sich zu sendende Dateien im zentralen Verzeichnis befinden müssen. Weiterhin muss `rvsbat` auf die zentrale Datenbank zugreifen können. Mehr über `rvsbat`,

insbesondere über die Befehle SEND, RESENTR und SENDJOB lesen Sie bitte in den Kapiteln 10.6, 10.7 und 10.8.

**Hinweis**: Der Begriff rvs® standalone wird als Gegensatz zu rvs® Data Center benutzt und bedeutet ein einzelnes rvs® (z.B. rvsXP oder rvsX).

Über Skripte kann eine Nachverarbeitung (RE oder JS) angestoßen oder rvs® Data Center gestartet bzw. gestoppt werden.

Über die grafische Benutzeroberfläche des rvs® Clients sind folgende Aktionen möglich:

- Konfiguration des rvs® Data Centers im laufenden Betrieb,
- Job-, Stations- und Benutzerverwaltung,
- Anzeige von Log-Meldungen und Statistik-Informationen mit Filtermöglichkeiten
- Snapshot-Anzeige der rvs® Data Center-Konfiguration.

### 2.11.5 Neue Systemkomponenten von rvs® Data Center

Folgende neue Komponenten unterscheiden rvs® Data Center von rvs® standalone.

#### 2.11.5.1 Ausfallsicherheit

Alle rvs®-Komponenten (rvscom, rvsmon, rvsxmt, ...) arbeiten auf allen rvs®-Knoten. Jeder Monitor (rvsmon) überwacht alle von ihm abhängigen Komponenten auf dem eigenen Knoten und die Monitore auf anderen rvs®-Knoten. Die Aktivitäten aller Knoten werden in der zentralen Datenbank in eine Datenbanktabelle protokolliert. Beim Ausfall einer Komponente auf einem rvs®-Knoten wird ein neuer Start dieser Komponente auf dem jeweiligen Knoten veranlasst und der abgebrochene Job zu Ende geführt. Im Falle des Ausfalls eines Monitors (was durch die Überprüfung der Aktivitäten in der Datenbanktabelle von einem anderen Monitor festgestellt wird) wird rvs® auf diesem Knoten neu gestartet. Der Neustart wird von demjenigen Monitor durchgeführt, der die fehlenden Aktivitäten in der Datenbank festgestellt hat.

Die zentrale Datenbank und die zentralen Verzeichnisse werden nicht überwacht. Wenn eines oder beides nicht zur Verfügung steht, kann das rvs® Data Center ihre Arbeit nicht mehr verrichten. Über ein konfigurierbares Zeitintervall lässt sich einstellen, wie lange jeder Monitor probiert, die ausgefallene Komponente zu erreichen. Wenn das Zeitintervall überschritten wurde, wird vom Monitor ein Error-Skript ausgeführt, über das der Fehler weiterkommuniziert werden kann.

**Hinweis**: Auf allen Knoten des rvs® Data Centers muss die Zeit synchronisiert sein, damit die Überwachung der Monitore korrekt arbeitet und die Log-Meldungen verfolgt werden können. Die Zeitsynchronisation ist eine Anforderung an das System, auf dem rvs® Data Center installiert ist.

Folgende neue globale Parameter dienen zur Konfiguration des Überwachungsmechanismus: `MONTIMEOUT`, `COMTIMEOUT`, `CNTMA`, `CNTGC` und `RECERREX`.

Neu sind auch folgende Parameter, die in der rvs®-Umgebungsdatei `rvsenv.dat` zu konfigurieren sind: `RVSNODENAME`, `SPERRTO`, `LOGINDB`, `LOGFORMAT`, `DBDL` und `DBTO`.

Über die Konfiguration der neuen rvs® Data Center-Parameter können Sie im Benutzerhandbuch rvsX, Kapitel 15 nachlesen.

#### 2.11.5.2 **Lastverteilung**

Jeder rvs®-Knoten kann alle Aufgaben übernehmen, und deshalb kann die Last auf alle Knoten gleichmäßig verteilt werden. Der Monitor eines Knotens, auf dem die Last gering ist, beginnt eher mit der Abarbeitung eines neuen Jobs, als ein Monitor, der unter hoher Last steht, da er eher auf die Datenbank zugreift, um zu überprüfen, ob neue Jobs abgearbeitet werden können (first come first serve).

**Hinweis**: Die dem rvs® Data Center vorgeschaltete Hardware (z.B. Brick für ISDN oder switch für TCP/IP) ist ebenfalls für die Lastverteilung von eingehenden Verbindungen, die auf die Knoten verteilt werden, relevant, nicht aber Bestandteil des rvs® Data Centers.

#### 2.11.5.3 Skalierbarkeit

Neue rvs® Knoten können rvs® Data Center hinzugefügt oder alte entfernt werden. Durch diese Skalierbarkeit der Anzahl der Knoten ist ein rvs® Data Center in der Lage, eine wesentlich höhere Menge an Daten und Aufträgen abzuarbeiten als ein rvs® standalone. Über das Hinzufügen oder Entfernen weiterer Knoten im laufenden Betrieb kann die Verarbeitungsmenge dynamisch angepasst werden, ohne dass rvs® Data Center gestoppt werden muss. Die Anzahl der sinnvoll verwendbaren Knoten ist abhängig von der verwendeten Umgebung (Rechner, Bandbreite des Netzwerks, Dateisystem, Datenbank, etc.).

#### 2.11.5.4 Log-Meldungen

Neben der schon bestehenden Möglichkeit, dass die Log-Meldungen in die Log-Datei (`rlog.log`, `rlco.log`) geschrieben werden, gibt es für rvs® Data Center noch die Möglichkeit, die Log-Meldungen von allen rvs®-Knoten direkt in die Datenbank zu schreiben. Die Reihenfolge der Log-Meldungen wird durch einen Prozesstyp und eine ProzessID sichergestellt.

**Beispiel (Log-Meldung):**

```
O: 2004/12/15 15:23:28{node1}[C49944][S0000000856]<CONNECT_IND      > Empfaenger: Verbindung
mit Station 'FRC10' mit Credit=99, Odette Buffer=2048, OFTP Komprimierung aufgebaut.
```

In diesem Beispiel wurde die Verbindung mit der Station FRC10 vom rvs®-Knoten node1 am 15.12.2004 um 15:23:28 aufgebaut. Im [C49944] ist C der Prozesstyp und 49944 die ProzessID. C steht für den rvs®-Kommunikationsprozess (rvscom).

**Hinweis**: Eine detaillierte Syntax-Erklärung von Log-Meldungen finden Sie im rvs®-Handbuch „Meldungen und Return-Codes“.

Die Auswertung der Log-Meldungen erfolgt über rvs® Client/Server, der die Log-Meldungen über Filter aus der Datenbank lesen kann (Fenster Admin -> Log Messages) oder über externe Applikationen, die die benötigten Daten direkt aus den entsprechenden Datenbanktabellen lesen. Über ein Datenbankskript (export_lt.sh) können Log-Meldungen aus der Datenbank in eine Datei exportiert werden.

#### 2.11.5.5 Parameteränderungen im laufenden Betrieb

Folgende Parameter können über rvsbat oder rvs® Client/Server mit dem Befehl setparm im laufenden Betrieb geändert werden: ODTRACLVL, LITRACELVL, SIDTRACE, STATISTICS, CMDDELETE, DTCONN1-20, TCPIPRCV, MAXX25RCV, OCREVAL und OEXBUF.

Für die Änderung aller anderen Parameter ist ein Stoppen und Starten des rvs® Data Centers notwendig, da sie auf allen Knoten gleich sein müssen.

## 2.12 Dateinamen

Die Syntax für gültige Dateinamen ist vom Betriebssystem abhängig. "Gleiche" Dateinamen können angegeben sein als:

| | |
|---|---|
| /myid/rechnung.dat | unter UNIX |
| c:\myid\rechnung.dat | unter Windows-Systemen |
| MYID/DATA(RECHNUNG) | unter OS/400 auf einer OS/400 |
| MYID.RECHNUNG.DATA | Unter MVS auf einem IBM Host |

. . .

Diese Namen sind auf manchen Systemen sensibel für Groß-/Kleinschreibung (z.B. UNIX), die Groß-/Kleinschreibung muss also beachtet werden. Sicherheitssysteme (wie RACF unter MVS) können weitere Einschränkungen bezüglich der auf einzelnen lokalen Systemen gültigen Dateinamen erzwingen.

Diese Unterschiede können nachfolgend beschriebene Probleme oder Mühen verursachen, wenn eine Datei an ein anderes Betriebssystem gesandt werden soll.

Hinweis: In der englischen Ausgabe dieses Handbuchs werden die englischen Begriffe `file` und `data set` synonym verwendet. In der vorliegenden deutschen Fassung wird durchgehend der Begriff `Datei` verwendet.

### Virtual Data Set Name

Für die Übertragung und Zustellung wird eine Datei durch ihren virtuellen Dateinamen (**VDSN**)[1] identifiziert. Sie können diesen **VDSN** im Feld **NEW DSNAME** (`Neuer Dateiname`) angeben, wenn Sie einen Sendauftrag erzeugen oder im Parameter **DSNNEW** des `SEND/CREATE` Kommandos (siehe 10.6). Wenn Sie keinen **VDSN** angeben, verwendet rvs® den Namen der zu sendenden Datei zur Erzeugung eines **VDSN**.

Wenn eine übertragende Datei ihrem Empfänger zugestellt wird, verwendet rvs® den **VDSN** als ein Kriterium bei der Suche nach übereinstimmenden residenten Empfangseinträgen. Wenn kein Eintrag vorhanden ist oder der am besten übereinstimmende Eintrag keinen Dateinamen definiert, benutzt rvs® den **VDSN** auch zur Erzeugung eines lokalen Dateinamens, unter dem die Datei gespeichert wird:

| | |
|---|---|
| rvsMVS | Verwendet den VDSN wie er ist, geben Sie deshalb beim Senden an einen MVS Host-Rechner unbedingt einen VDSN an, der den MVS Namens-vereinbarungen entspricht und mit einem High-Level-Begrenzungszeichen beginnt, das zu RACF passt. Verwenden Sie keine Anführungszeichen (″) oder Apostrophe (') für die Begrenzung Ihres Dateinamens. |
| portable rvs | Verwendet den VDSN zur Erzeugung eines Namens für die einzelne Datei; der Pfad- oder Bibliotheks-namen, wo die Datei gespeichert wird, wird der lokalen rvs® -Umgebung entnommen (für weitere Information zur rvs® Konfiguration sprechen Sie bitte mit Ihrem rvs® Administrator oder lesen Sie im "rvs® Benutzerhandbuch" nach). |

Der **VDSN** ist auch ein Kriterium für die Angabe eines Auftrages, der nach einem Sendeversuch ausgeführt werden soll (für weitere Informationen siehe Benutzerhandbuch "Job-Start nach Sendeversuch").

[1] VDSNs werden bei ODETTE File Transfer Protokoll benutzt, um den Dateinamen an den nächsten Knoten zu übergeben. Bei der Generierung eines Standard-VDSN richten sich die maximale Länge (26 Zeichen) und der Zeichensatz nach dem ODETTE Protokoll.

## Zeitstempel

Wenn Zeitstempel angefordert werden, erzeugt rvs® bei der Lieferung einer Datei eindeutige Dateinamen, indem es eine Belegnummer anhängt oder den letzten Teil des Dateinamens durch einen numerischen Wert ersetzt. Für Dateien mit im übrigen Teil identischen Namen gibt diese Nummer die Reihenfolge der Dateianlieferung an (wenn nicht eine oder mehrere alte Dateien gelöscht wurden, verwendet rvs® die kleinste verfügbare Nummer). Die aktuell von rvs® unterstützten Dateisysteme erlauben keine Addition des Echtzeit-Zeitstempels zum Dateinamen, d.h. Datum und Zeit der Lieferung.

Zeitstempelung kann in einem residenten Empfangseintrag angefordert werden. Sie wird ausgeführt, wenn die Datei zugestellt wird.

# 3 Protokollschichten in der rvs® Kommunikation

## Allgemeiner Überblick

Die funktionale Hierarchie der Protokollschichten in einer rvs® Umgebung können als Unterschichten der Anwendungsschicht (Schicht 7) des OSI-Referenzmodells betrachtet werden. Der rvs® Leitungstreiber und die Komponenten des Kommunikationssystems, auf dem er aufgebaut ist, wie X.25, SNA LU6.2 oder TCP/IP, gehören zu den unteren Schichten. rvs® ist jedoch nicht mit den ISO/OSI Standards kompatibel, sondern folgt lediglich den Empfehlungen des Odette Gremiums. Die folgende Tabelle gibt einen vereinfachten Überblick über die funktionale Hierarchie, z.B. mit LU 6.2 Kommunikation:

| LOKALES SYSTEM | | ENTFERNTES SYSTEM |
|---|---|---|
| Benutzer, Bediener,<br>Programme, Arbeitsaufgaben | | Benutzer, Bediener,<br>Programme, Arbeitsaufgaben |
| I. ! | | ! |
| 0 ! | | ! |
| MONITOR → Abschicken der lokalen Arbeitsaufgabe | | MONITOR → Abschicken der lokalen Arbeitsaufgabe |
| 1 ! | | ! |
| MASTERTRANSMITTER | | MASTERTRANSMITTER |
| 2 ! | | ! |
| (lokale) Daten ↔ KOMMUN. PGM. | | (lokale) Daten ↔ KOMMUN. PGM. |
| 3 ! | | ! |
| OFTP | -------OFTP Session------- | OFTP |
| ! | | ! |
| LINEDRIVER | -----Leitungstreiber Session --- | LINEDRIVER |
| ! | | ! |
| ! | (LU 6.2 conversation) | ! |
| NETZWERK | ------Netzwerk Session ---- | NETZWERK |
| ! | | ! |
| ! | (SNA Session) | ! |

-------------------Physikalisches Netzwerk-------------------

## 3.1 Netzwerk

rvs® kümmert sich nicht um die physikalischen Einzelheiten des Netzwerks, z.B. ob X.25, gemietete Leitungen oder Token Ring benutzt werden. Die Steuerung und die Definitionen des physikalischen Netzwerks erfolgen extern vom rvs® portabel. Das Vorhandensein von LU 6.2 Diensten und PU 2.1 Support sind bei SNA Netzwerken für beiden Kommunikationspartner erforderlich.

## 3.2 rvs® Leitungstreiber

Der rvs® Leitungstreiber dient zusammen mit den entsprechenden Systemkomponenten dazu, eine End-zu-End-Verbindung herzustellen (X.25 Verbindung, LU 6.2 Konversation oder TCP/IP Verbindung), die die sichere und nachvollziehbare Übertragung von Dateien von Stationen zu Stationen ermöglicht. Er kommuniziert mit dem Leitungstreiber der anderen rvs® Station auf der Basis eines speziellen Leitungstreiberprotokolls, das vom Netzwerktyp, z.B. X.25, LU 6.2 oder TCP/IP, abhängt. Der rvs® Leitungstreiber bekommt von der OFTP-Schicht Anfragen, eine Verbindung mit dem Netzwerk herzustellen oder abzubrechen. Nach der erfolgreichen Verbindung, bekommt er die Dateien vom entfernten Leitungstreiber und leitet sie an die lokale OFTP Schicht weiter und umgekehrt.

## 3.3 OFTP

Der Zweck des Odette Übertragungsprotokolls (OFTP) ist die zuverlässige Übertragung von Dateien. Das OFTP eröffnet eine Protokollsitzung mit dem OFTP der entfernten Station, die logisch über der Leitungstreiberverbindung läuft.

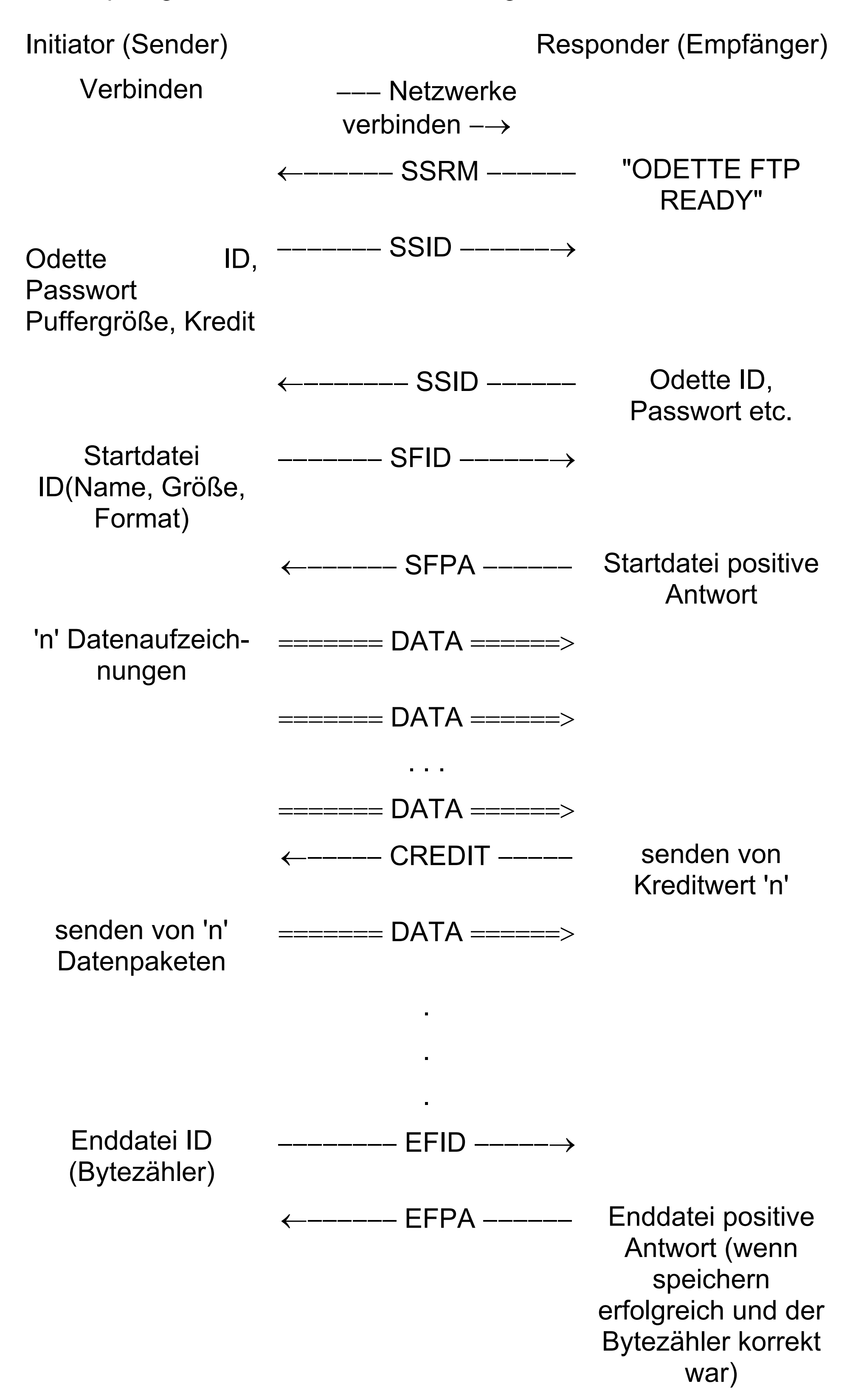
Nach dem Beginn der OFTP Sitzung tauschen beide Seiten ihre IDs und Passwörter aus, verhandeln über Parameter, wie die Größe des Odette Austauschpuffers, den Odette Kreditwert (die Anzahl der Puffer, die die sendende Seite ohne eine Antwort abzuwarten senden kann). Sie tauschen Informationen über den Namen, die ungefähre Größe und das Format der zu übertragenden Datei aus.

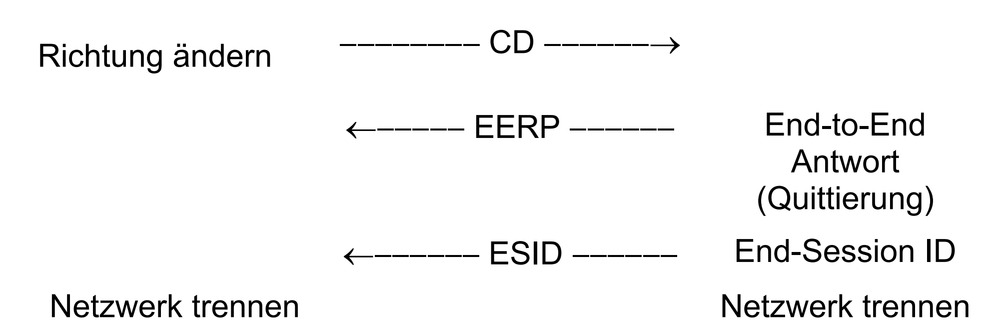
Während der Übertragung findet eine Kompression und Dekompression der Daten statt. Nach der Übertragung werden die Bytezähler auf beiden Seiten verglichen. Nach der erfolgreichen Speicherung der Datei wird eine Quittierung an die Ausgangsstation gesandt. Falls die Übertragung gestört wurde, z.B. durch Leitungsunterbrechung, verfügt das OFTP Protokoll über einen Mechanismus, der die Wiederaufnahme der Übertragung an dem Unterbrechungspunkt gewährleistet.

Funktion `Richtungsänderung`: Wenn alle Dateien übertragen worden sind, wird der Empfänger vom Sender aufgefordert, eigene wartende Dateien zu senden.

Für mehr Protokolleinzelheiten konsultieren Sie bitte die Veröffentlichungen der Odette und VDA Gremien: "Odette Specifications for File Transfer".

In der Tabelle unten wird der allgemeine, aber vereinfacht dargestellte Nachrichtenfluß in einer Odette Sitzung gezeigt. Die sendende Seite ist der Initiator, die empfangende Seite arbeitet als Reagierender.






## 3.4 rvs® Kommunikationsprogramm

Wie das rvs® Kommunikationsprogramm die systemabhängige physikalische Datei Input/Output behandelt, wird am Anfang des OFTP geregelt. Wenn es notwendig ist, werden die Daten vom EBCDIC in ASCII Code übersetzt und umgekehrt. Weitere Informationen über Code-Umwandlung finden Sie im rvs® Benutzerhanbuch. Das Kommunikationsprogramm speichert eingehende Informationen in einer temporären Datei, wenn es im Empfangsmodus arbeitet. Auch Datenformate, die vom lokalen System nicht unterstützt werden, wie z.B. Dateien mit festen oder variablen Datensätzen unter OS/2, können gespeichert und weiterverschickt werden; sie werden nach dem Empfang in ein unterstütztes Format konvertiert.

# 4 LU 6.2 Basiskonzept

Das Ziel dieses Kapitels ist, einen kurzen Überblick über die Grundfunktionen der LU 6.2 Kommunikation zu geben, und das Verstehen der entsprechenden Parameter der rvs® Stationstabelle zu erleichtern. Einzelheiten über die Anbindung und ihre Spezialitäten und Beschränkungen bei bestimmten Systemplattformen werden im entsprechenden Installationshandbuch behandelt. Wenn Sie mehr Einzelheiten über LU 6.2 erfahren wollen, konsultieren Sie bitte die IBM Veröffentlichungen und/oder die LU 6.2 Dokumentation Ihres Systemanbieters. Einige nützliche IBM Publikationen sind nachfolgend gelistet:

- Systems Application Architecture, Communications Reference, SC26-4399
- System Network Architecture, Concepts and Products, GC30-3072-4
- System Network Architecture, Sessions Between Logical Units, GC20-1868-2
- System Network Architecture, Type 2.1 Node Reference, SC30-3422-2
- Überblickskapitel in systemspezifiscen APPC Handbüchern, wie
  - OS/2 (EE 1.3) APPC Programming Reference
  - OS/400 APPC Programmer's Guide
  - RS/6000 APPC Programmer's Guide

## 4.1 Warum LU 6.2 Kommunikation

Die wichtigsten Gründe für die Wahl von LU 6.2 (SNA Logical Unit type 6.2) Kommunikationstechnik als dem zukünftigen Hauptmittel für alle rvs® Versionen sind:

- Eine hochentwickelte Programm-zu-Programm Kommunikation (APPC) Anwendungssschnittstelle für LU 6.2 ist für fast alle modernen Computersysteme verfügbar.
- Die APPC Programmschnittstelle ist standardisiert, mindestens auf der funktionalen Ebene. So können Anwendungsprogramme, die APPC benutzen, ohne große Änderungen an andere Systeme angepaßt werden.
- Die Anwendungsschnittstelle ist von den physikalischen Eigenschaften des Netzwerkes unabhängig. Die Anwendung sucht sich lediglich eine verfügbare SNA Session, die sie nutzen kann.
- Die Steuerung der Netzwerkressourcen durch das Anwendungsprogramm ist in der Regel nicht notwendig und meistens bei APPC auch nicht möglich. Sie muß extern mit Hilfe der üblichen Netzwerk Tools des Systems ausgeführt werden.

Die Alternative innerhalb der SNA-Welt wäre, die veraltete LU 0 Schnittstelle einzusetzen. LU 0 wird von vielen Computer-Systemen nicht unterstützt und wenn doch, ist die Programmschnittstelle in großem Maße vom entsprechenden System abhängig. Zusätzlich, verursacht sie Schwierigkeiten in der Systemsteuerung.

## 4.2 LU 6.2 Basisfunktionen

Das SNA Netzwerk und die APPC Anwendungsschnittstelle müssen als unabhängig voneinander betrachtet werden. Im Rahmen des OSI-Referenzmodells gehört APPC zur Schicht 7, während das Netzwerk (grob gesehen) die Schichten 0 bis 6 abdeckt.

Das SNA Netzwerk muß eine physikalische und logische Verbindung zwischen den Endpunkten der Kommunikation bereitstellen, zwischen den LUs 6.2, die in der Regel reale Computer-Systeme sind, und als gleichwertige Partner im Netzwerk funktionieren. Diese Verbindung ist eine SNA Session, die in einem speziellen Log-Modus aufgebaut wird. Einzelheiten des physikalischen Netzwerks, z.B. ob Token-Ring, X.25 oder SDLC Links benutzt werden, müssen auf der Definitionsebene der entsprechenden Grundkommunikationsressourcen wie VTAM, NCP, etc. behandelt werden.

Die LU 6.2 Anwendung, in diesem Fall rvs®, benutzt eine verfügbare SNA Session, um eine logische Kommunikationsverbindung zwischen den Partnern aufzubauen. Diese logische Verbindung wird LU 6.2 Konversation genannt. Eine Konversation kann nur zwischen zwei Anwendungsprogrammen existieren, sie startet ein Partnerprogramm auf der entfernten Seite. Die lokale Anwendung braucht den LU-Namen des Partnersystems, den Namen des (Log)Modus, um eine unbenutzte Session mit den richtigen Eigenschaften auszuwählen, und den Namen des Transaktionsprogramms des entfernten Partners (TP-Name).

Der TP-Name in unserem Fall ist der Name des rvs® Kommunikationsprozesses: `rvscom`.

Nachdem die Konversation hergestellt worden ist, verläuft die Kommunikation zwischen den beiden Partnern mit Hilfe von APPC Verben, die mehr oder weniger standardisierte Funktionsaufforderungen darstellen wie `Send` (`Senden`) oder "ReceiveAndWait" (Empfangen und Warten). Die LU 6.2 Konversationen haben halbduplex, oszillierende Eigenschaften, was bedeutet, dass die Seiten nur abwechselnd und nie gleichzeitig die Kontrolle haben.

## 4.3 "Mapped" oder "Basic" Konversation

Abhängig von den Initialisierungsparametern kann die Konversation vom Typ `basic` oder `mapped` sein und die Sychronisationsebenen `bestätigen` (`confirm`) oder `keine` (`none`) aufweisen.

Der Hauptunterschied zwischen der Basiskonversation und der abgebildeten Konversation ist, dass in der Basiskonversation die Längeninformation über den zu versendenden Block von Benutzerdaten ein Teil dieser Daten ist. Die Anwendung muß beim Senden die Informationslänge zu den Daten hinzufügen und sie beim Empfang aus den Daten entnehmen. Zurzeit unterstützt rvs® nur abgebildete Konversationen, d.h. das System akzeptiert und bestätigt die Längeninformation durch einen Parameter der Sende- bzw. Empfangsaufforderung.

Die Synchronisationsebene Bestätigen (confirm) bedeutet, dass zum Zweck des Abgleichs der eine Partner den anderen auffordern kann, ein Bestätigungssignal zu schicken, und dass die Konversation nicht vor dem Empfang dieser Bestätigung fortgeführt wird. rvs® unterstützt beide Modi. Das Benutzen der Synchronisationsebene Bestätigen (confirm) macht den Prozeß sicherer, aber auch langsamer.

## 4.4 Sicherheit

Die meisten Systeme bieten die Möglichkeit, Sicherheitsinformationen beim Verbindungsaufbau zu übermitteln. Die Sicherheitsinformation besteht aus einer Benutzer-ID und einem Passwort, das für das entferntes System gültig ist. Beide sind notwendig, um ein entferntes Transaktionsprogramm zu starten. Einige Systeme unterstützen Standardbenutzer und prüfen keine Sicherheitsmaßnahmen, wenn ein solcher Standardbenutzer definiert worden ist. rvs® bietet für die Sicherheitsinformationen eine entsprechende Stationstabelle.

## 4.5 Abhängige und unabhängige LUs

Es gibt zwei Arten von LU 6.2 logischen Einheiten: unabhängige LUs (wirkliche Partner) und abhängige LUs. Die letzteren sind LUs, die vom Steuerpunkt der Systemdienste SSCP gesteuert werden, z.B. VTAM bei einem SNA-Host. Abhängige LUs werden als lokale Ressourcen (wie ein Terminal) behandelt, die zu einer physikalischen Einheit des Typs 2.0 (PU 2.0) gehören.

Unabhängige oder Partner-LUs erfordern die Unterstützung physikalischer Einheiten des Typs 2.1, (PU 2.1). Auf Nicht-Großrechner-Systemen sind die unabhängige LU und die PU 2.1 das System selbst. Die unabhängigen LUs und die entsprechenden physikalischen Einheiten des Typs 2.1 sind immer als aktiv (active) zu betrachten, und sie akzeptieren niemals SSPC Kommandos wie actpu (Aktiviere physikalische Einheit) oder actlu (Aktiviere logische Einheit) von außerhalb. Das bedeutet auch, dass sie nicht von außerhalb deaktiviert werden können. Bevor eine SNA Session aufgebaut ist, z.B. ein Verbindungsbild ausgetauscht ist, tauschen beide Seiten ihre XIDs. Das geschieht auch dann, wenn die Verbindung über gemietete Leitungen erfolgt, weil der XID-Tausch zum Verhandeln der Stationsrollen (primäre oder sekundäre) beider Partner dient.

rvs® ist für die Nutzung mit unabhängigen LUs und PU 2.1. bestimmt. Ein Grundvorteil von PU 2.1. ist, dass sie parallele Sessions unterstützt. Das bedeutet, dass alle Partner-LUs freien Zugriff auf die lokale LU haben, und dass diese im Gegenzug zeitgleich auf sie zugreifen kann. Mit unabhängigen LUs kann rvs® gleichzeitig viele Konversationen eröffnen und Daten parallel über dieselbe physikalische Verbindung übertragen.

Die Benutzung von abhängigen LUs wurde auf rvs® noch nicht getestet und wird deshalb als "nicht von rvs® unterstützt" bezeichnet. Bei abhängigen LUs wird der Beginn einer Session vom SNA-Host gesteuert. Nur eine einzige Session pro LU ist

möglich. Mit einer solchen Verbindung kann rvs® nur ein Datenpaket in dem gegebenen Zeitabschnitt übermitteln.

Während die meisten Nicht-Großrechner-Systeme PU 2.1 als benutzerfreundlichen Standard unterstützen, erfordert es Wissen und Einsatz, um die Unterstützung von PU 2.1 bei SNA-Hosts zu gewährleisten.

### 4.6 Auswirkungen von LU 6.2 auf das rvs® Design

Die Grundeigenschaften von LU 6.2 haben das speziell angepaßte Design von rvs® bewirkt, das sich von der rvsMVS Anbindung unterscheidet. In Nicht-Großrechner-Systemen deckt sich die LU mit dem System, und das TP, der rvs® Empfänger, startet in unserem Fall als unabhängiger Hauptprozeß.

Einige LU 6.2 Anbindungen (z.B. unter SINIX) erfordern, dass jede APPC Anwendung als unabhängiger Task läuft. Zum Zweck der Symmetrie und der Mobilität wurden der Sender und der Empfänger in einem einzigartigen Kommunikationsprogramm vereint, das beide Rollen annehmen kann. Das Kommunikationsprogramm wurde als unabhängiger Task oder Prozeß geschaffen.

Der rvs® MasterTransmitter hat die Aufgabe, immer wenn ein Datenpaket zur Übertragung bereitsteht, einen unabhängigen Kommunikationsprozeß zu starten. Der MasterTransmitter und der Kommunikationsprozeß kommunizieren über ein System von abhängigen Semaphoren und über die rvs® Datenbank.

Das ist der Unterschied zu der rvsMVS Anbindung, wo alle LU 6.2 Aufgaben als Unteraufgaben des rvs® Monitors laufen. Bei rvsMVS ist die LU die rvs® Unteraufgabe der LU 6.2-Steuerung, während das TP eine ihrer Unteraufgaben darstellt.

# 5 X.25 Native Kommunikation

### Eigenschaften von X.25

Das Ziel dieses Kapitels ist, einen allgemeinen Überblick über die X.25 Kommunikationstechniken und ihre Anwendung bei rvs® zu vermitteln. Die Unterschiede in der Anbindung und ihre Grenzen bei speziellen Plattformen werden im entsprechenden Installationshandbuch behandelt.

X.25 native Kommunikation ist die Grundlage für `offene` Kommunikation nach der Definition in dem Odette Übertragungsprotokoll. `Offen` bedeutet eine weltweite Verbindungsfähigkeit zwischen Stationen, die OFTP kompatible Kommunikationsprodukte wie rvs® betreiben, indem sie öffentliche Datenpaketvermittlungswählnetze nutzen. X.25 Netzwerke basieren auf dem OSI-Referenzmodell und decken die OSI-Schichten 1 bis 3 ab. Jede Installation, die mit einem öffentlichen X.25 Netzwerk verbunden ist, kann über eine eindeutige X.25 Adresse von bis zu 15 Ziffern erreicht oder kontaktiert werden. So wird eine fast telefonähnliche Verbindung erreicht. Nicht alle nationalen X.25 Netzwerke arbeiten jedoch auf derselben funktionalen Ebene, so dass funktionale Begrenzungen auftreten können.

Das physikalische Routing und der Transport der Daten, die in Datenpaketen von 128 Byte organisiert sind, wird bei einem X.25 Netzwerk vom öffentlichen Netzwerkanbieter durchgeführt. Es ist möglich, dass Datenpakete, die zu demselben Block von Benutzerdaten gehören, über unterschiedlichen Routen transportiert werden. Das Netzwerk wird sie jedoch in der richtigen Reihenfolge an das Endziel liefern. Das Netzwerk sorgt für automatische Geschwindigkeitskontrolle, was die Kommunikation zwischen Kommunikationspartnern mit unterschiedlichen Übertragungsgeschwindigkeiten ermöglicht. Spezielle X.25 Protokollelemente (RR, RNR) steuern das End-zu-End-Tempo, um ein Datendurcheinander zu verhindern.

Die End-zu-End-Verbindung in einem X.25 Netzwerk wird ein virtueller Kanal genannt, VC (virtual circuit). Ein physikalischer Link zum X.25 Netzwerk kann bis zu 255 VCs tragen, abhängig von Ihrem Vertrag mit dem Anbieter. Wegen dieser Fähigkeiten wird der Link ins paketwechselnde Netzwerk auch Mehrfachkanal-Link genannt.

Es gibt zwei Arten virtueller Kanäle:

- SVCs, wechselnde virtuelle Kanäle,
- PVCs, permanente virtuelle Kanäle.

Die PVCs verhalten sich wie gemietete Leitungen und stellen ein anderes Punkt-zu-Punkt-Verhältnis dar. SVCs haben auch einen Punkt-zu-Punkt-Charakter, das eine Ende jedoch kann bei der Installation frei gewählt werden. Am Ende einer logischen

Verbindung wird der SVC unterbrochen. rvs® ist für den Betrieb auf der Basis von SVCs bestimmt.

Kunden, die an ein X.25 Netzwerk angebunden werden, müssen die Anzahl der PVCs und/oder SVCs bestimmen, für die ihr Mehrfachkanal-Link konfiguriert werden soll. Die Anzahl der Kanäle ist die obere Grenze der möglichen parallelen Verbindungen. Andererseits ist die Zahl der virtuellen Kreise eine Kostenfrage. Die rvs® Anforderungen an die Anzahl der virtuellen Kanäle hängen von der erwarteten Belastung ab, genauer von der Anzahl der Datenpakete, die gleichzeitig parallel übertragen werden sollen.

Als minimale Anforderung für einen sehr schwachen Verkehr sollten 2 VCs vorhanden sein. In der Regel reichen ca. 10 VCs aus.

Abhängig vom Anbieter bieten X.25 Netzwerke eine Reihe von Zusatzfunktionen wie

- umgekehrte Belastung
- geschlossene Benutzergruppen
- schnelle Selektion
- Lieferbestätigung
- Benutzerdaten

und viele andere. Die meisten von ihnen erfordern das beidseitige Einverständnis der Partner.

Die meisten X.25 Netzwerke unterstützen eine Funktion, genannt Paket Assembler Disassembler, PAD, (Paket Zusammen- und Auseinandersetzung). Ein PAD ist eine Hardware oder Softwarekomponente, die einfache Standard-ASCII-Datenströme in X.25 Pakete konvertiert und umgekehrt.

Ein PAD kann nur von der ASCII Seite angerufen werden. Auf X.25 Seite kann es nur anrufen. Eine Standard PAD Konfiguration ist ein ASCII Terminal oder ein PC, der ein öffentliches PAD über ein Telefonmodem anruft, um Zugang zu einem entfernten Host über das öffentliche X.25 Netzwerk zu erlangen.

Das Standardprotokoll für die ASCII Seite ist X.28, die vom Benutzer konfigurierbaren PAD-Parameter sind als X.3 standardisiert, und das End-zu-End-Verbindungsprotokoll in diesem Kontext als X.29. Die Fähigkeit, PAD-Verbindungen zu unterstützen, wird in einer der nächsten Versionen von rvs® portabel verfügbar werden.

Anstelle ein teures öffentliches X.25 Netzwerk wie DATEX-P zu benutzen, ist es möglich, einen Adapter zwischen dem X.25 Bord und dem ISDN Port zu installieren. Der Adapter übersetzt X.25 Pakete für das ISDN und umgekehrt. Dieses Protokoll ist als X.31 standardisiert. Beide Kommunikationspartner brauchen dann einen Adapter. Wenn Sie Fragen haben, wenden Sie sich bitte an Ihren Vertriebspartner.

## 5.1 Verbindungsaufbau X.25

Um einen rvs® Partner über X.25 anzurufen, sind einfach die X.25 Rufadresse und optional die X.25 Benutzerdaten erforderlich.

Der Verbindungsaufbau auf der X.25 Netzwerkebene beginnt mit dem Senden eines X.25 Paketes, welches aus der Zieladresse, der eigenen Adresse und den Benutzerdaten besteht. Wenn die andere Seite das Anrufdatenpaket zurückschickt, ist die Verbindung zustande gekommen, und ein SVC ist aktiviert worden. Falls die entfernte Station den Anruf nicht erwartet, oder irgendeiner der Parameter nicht stimmt, wird sie den Anruf mit einem leeren Paket beantworten, was auch den SVC unterbricht.

Ein erfolgreicher Verbindungsaufbau erfordert das Aktivieren eines rvs® `Listening`-Tasks auf der Seite, die den Anruf bekommt. Bei einigen Systemen, wie IBM-Großrechnern, aktiviert rvs® einen Antwort-Task nach dem Empfang eines X.25 Anrufs. Bei anderen, wie den meisten portablen rvs® Plattformen, müssen ein oder mehrere aktive Empfänger voraktiviert sein, um die eingehenden Anrufe entgegenzunehmen.

Wenn einmal ein SVC aufgebaut wurde, können beide Seiten frei Datenpuffer in vollem Duplexmodus austauschen. Die einzige Sicherheitsüberprüfung zur Zeit ist die Überprüfung der Anruferadresse seitens der angerufenen Station. Größere Sicherheit bietet das von rvs® benutzte Odette Protokoll, das den Austausch von SSIDs mit Stations-IDs und Passwörtern voraussetzt. Wenn beide Seiten übereinstimmen, wird eine Odette Session aufgebaut, und die Dateiübertragung kann beginnen.

Unter normalen Bedingungen braucht rvs® keine Anruferdaten. Wenn jedoch andere Anwendungen denselben X.25 Mehrfachkanal nutzen, müssen die Anruferdaten benutzt werden, um den eingehenden Anruf zu der richtigen Anwendung zu routen. Das erste Byte der Anruferdaten wird in der Regel als die so genannte Protokollidentifizierung, PID, interpretiert. Einige Zeichen sind für PID reserviert, z.B. X'C3 und X'C4 für SNA, X'01 für PAD, X'C0 für ASYNC etc. Im Normalfall erwartet rvs® gar keine PID. X'01 PIDs sind in Anrufen zu finden, die von PADs stammen, während X'C0 PIDs von einfacheren OS/400 Produkten stammen. Diese PIDs können von spezieller Bedeutung für rvs® sein.

## 5.2 Auswirkungen von X.25 auf rvs® Design

Die Notwendigkeit, einer besseren Kontrolle der Anzahl der gleichzeitig aktiven Übertragungsprogramme innerhalb der internen X.25 Kommunikation wurde immer deutlicher, seitdem die Kunden aus Kostengründen ihre Mehrfachkanäle mit einer sehr geringen Zahl virtueller Kanäle benutzen. Als Reaktion darauf wurde rvs® um den MasterTransmitter erweitert. Um mit X.28 Kommunikationspartnern umzugehen, die sich über X.25 PAD einwählen, wurden die vorher separaten `Nur-Senden-` und

`Nur-Empfangen-Module` in ein einzigartiges Kommunikations-Programm integriert, das beide Richtungen bearbeiten kann. Das ist notwendig, weil PAD-Verbindungen nur eine einzige Session haben, die zusätzlich nur von der X.28 Seite initiiert werden kann.

rvs® unterstützt mehrfache X.25 Leitungen. Wenn mehr als eine X.25 Leitung benutzt wird, braucht rvs® für jede Leitung einen Datenbankeintrag (XP) (mit Adresse, Link, alias Name usw.).

# 6 TCP/IP und rvs®

Dieses Kapitel beschreibt die TCP/IP Anwendungs-Schnittstelle, das Adressieren bei TCP/IP sowie den Aufbau der Verbindung mit rvs®.

## 6.1 TCP/IP Anwendungs-Schnittstelle

rvs® kümmert sich nicht darum, wie und auf welchem Weg die Daten innerhalb des TCP/IP Netzwerks transportiert werden. rvs® benutzt eine Anwendungs-Schnittstelle auf ziemlich hohem Niveau, die aus den wenigen Grundfunktionen genannt: `connect` (verbinden), `listen` (hören), `accept` (akzeptieren), `send` (senden), `receive` (empfangen) und `close` (schließen) besteht, die in der allgemein zugänglichen C-Socket-Bibliothek zu finden sind. Die C-Socket-Bibliothek sorgt für eine zuverlässige Übertragung der Datenblöcke.

Die Kommunikation über TCP/IP setzt voraus, dass der eine Partner, der die Verbindung initiiert, als `Client`, und der andere Partner als `Server` agiert. Der Client-Prozeß, in unserem Fall der rvs® Sender, startet eine Verbindungsanfrage zu einer Adresse, wo er weiß, dass der Partner, der rvs® Empfänger, als Serverprozeß `hört`. Die Verbindung ist aufgebaut und fertig zur Benutzung, wenn der Serverprozeß eine `Zusage` (Accept) gesendet hat, was nach dem Empfang der Verbindungsanfrage des Clients erfolgt. Ein voller Duplexaustausch der Daten ist dann ab sofort möglich.

## 6.2 TCP/IP Adressierung

Das Adressieren in einem TCP/IP Netzwerk ist ganz einfach. Eine TCP/IP Adresse besteht aus zwei Teilen:

Internet Adresse — Das ist eine weltweit eindeutige Adresse, die ein Hostsystem definiert. Sie ist ein 32-Bit-Integer Wert, der manchmal in der Form `aaa.bbb.ccc.ddd` angegeben wird, wobei `aaa` bis `ddd` dreistellige Dezimalwerte sind, die von links (höhere Ordnung) nach rechts (niedrigere Ordnung) je eins der 4 Bytes der Internet-Adresse beschreiben.

Port-Nummer — Jeder Host im Netzwerk benutzt eine Anzahl von 64K Unteradressen, die Ports genannt werden. Jeder Port wird von einem 16 Bit-Integer Wert beschrieben. Die Port-Nummern von 0 bis 1023 sind reserviert.

Eine TCP/IP Verbindung wird immer genau zwischen einer lokalen und einer fernen `Internet.Port`-Kombination installiert, indem die eine Seite Client, die andere Seite Server ist. Eine `Internet.Port`-Kombination ist an einem sogenannten Socket gebunden, den das Anwendungsprogramm als Referenz benutzt. Während es dem Client freisteht, einen freien Port auszuwählen, muß der Server einem bestimmten Port zuhören, der dem Client bekannt ist. Sonst könnte der Client den Server nicht finden und würde sich unter Umständen sogar mit einer falschen Applikation verbinden.

# 7 XOT-Kommunikation

In diesem Kapitel bringen wir ein paar Hinweise für die Konfiguration von XOT-Router.

XOT(X.25 over TCP/IP)-Router sind in der Lage, X.25-Pakete zwischen einem TCP/IP-Netztwerk auf einer Seite und einem X.25 oder ISDN-Netzwerk auf der anderen Seite zu routen. Wie Sie rvs® für XOT konfigurieren sollen, lesen Sie bitte im Benutzerhandbuch für rvsXP oder rvsX.

## 7.1 Systemanforderungen

Um die XOT-Funktionalität in rvs® zu benutzen, brauchen Sie eine IP-Verbindung zu einem XOT-Router (z.B. CISCO 801, CISCO 2600 oder BINTEC X4300).

Die XOT-Funktionalität in rvs® ist im Moment für die folgenden Plattformen verfügbar:

- LINUX
- Windows.

Wir haben die XOT-Funktionaltät in rvs® mit folgenden Routern getestet:

- CISCO 801
- CISCO 2600 und
- BINTEC X4300.

## 7.2 CISCO-Konfiguration

Im nächsten Kapitel werden wir zuerst unser Testszenario beschreiben und ein paar Beispiele für die CISCO-Routerkonfiguration aufführen.

Um eine BRI-Schnittstelle für ausgehende und eingehende Richtung zu konfigurieren, verwenden wir dialer-Schnittstellen (dialer interfaces). Es muss mindestens ein dialer für die Richtung „Senden“ und ein dialer für die Richtung „Empfangen“ existieren.

Vor den XOT-Tests in rvs®, haben wir die X.25-Funktion mit PAD–Befehlen in CISCO-IOS überprüft.

### 7.2.1 CISCO 801

Da wir ein paar Probleme mit der firmware 12.2 hatten, wurde die firmware 12.3 eingesetzt.

### 7.2.2 CISCO 2600

Da wir ein paar Probleme mit der firmware 12.2 hatten, wurde die firmware 12.3 eingesetzt.

### 7.2.3 Links für mehr Information

Für weitere Informationen über XOT und CISCO lesen Sie bitte die CISCO-Handbücher. Hier sind ein paar interessante Links:

http://www.cisco.com/en/US/tech/tk713/tk730/tech_digests_list.html
http://www.cisco.com/en/US/products/sw/iosswrel/ps1818/products_configuration_guide_chapter09186a0080087858.html
http://www.cisco.com/en/US/tech/tk713/tk730/technologies_q_and_a_item09186a00800a3c0b.shtml
http://www.cisco.com/en/US/products/hw/routers/ps380/tsd_products_support_series_home.html
http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122relnt/800/rn800xi.htm#1075191
http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123relnt/800/rn800xa.htm

### 7.2.4 Beispiele für CISCO-Router-Konfiguration

In diesem Kapitel werden wir ein paar Hinweise und Beispiele für die Konfiguration von CISCO-Routern geben.

- Die ausgehende dialer-Schnittstelle (the outgoing dialer interface) muss mit **encapsulation x25 DTE** konfiguriert werden.
- Die eingehende dialer-Schnittstelle (incoming dialer interface) muss mit **encapsulation x25 DCE** konfiguriert werden.
- Es existiert ein idle Timer (Wartezeit-Timer) für eingehende ISDN-Verbindungen. Dieser Timer erkennt die X.25-Pakete über ISDN nicht, so dass die Möglichkeit besteht, dass die Verbindung, während der Übertragung, unterbrochen wird. Dieser Timer soll in der eingehenden BRI Schnittstelle auf einen hohen Wert gesetzt werden (z.B. **dialer idle-timeout 700000**).
- Routing für beide Richtungen muss konfiguriert werden. Jeder ausgehende DIALER repräsentiert eine Partnerstation mit ihrer eigenen ISDN-Nummer (`dialer string <ISDN-No>`). Der eingehende Dialer horcht auf seiner eigenen ISDN-Nummer (`dialer called <ISDN-No>`).
- Sie können die ankommenden XOT-Anrufe mit dem folgenden Eintrag routen: **x25 route [called x25-Address in the XOT-packet] interface DIALER[number]**
- Sie können die ankommenden ISDN/X.25-Anrufe mit dem folgenden Eintrag routen. Diese Rufe können Sie mit dem folgenden Eintrag routen: **x25 route [called x25-Address in the x25-packet] XOT [IP of the XOT-station]**

Hier sind ein paar Konfigurationsbeispiele für CISCO-Router.

**Beispiel**: CISCO 801:

```
version 12.3
service pad to-xot
service pad from-xot
service tcp-keepalives-in
service tcp-keepalives-out
!
hostname Receiver
!
isdn switch-type basic-net3
x25 routing
!
interface Ethernet0
 ip address [IP-ADDRESS]
 no cdp enable
!
interface BRI0
 no ip address
 encapsulation hdlc
 no ip mroute-cache
 dialer pool-member 1
 dialer idle-timeout 1
 isdn switch-type basic-net3
 isdn point-to-point-setup
 no fair-queue
 no cdp enable
!

interface Dialer1
 description outgoing ISDN calls
 no ip address
 encapsulation x25
 no ip mroute-cache
 dialer pool 1
 dialer idle-timeout 1
 dialer string <partner-ISDN-number 1>
 dialer max-call 1
 dialer-group 1
 x25 htc 1
 x25 win 7
 x25 wout 7
 no cdp enable


interface Dialer10
 description incoming ISDN calls
 no ip address
 encapsulation x25 dce
 no ip mroute-cache
 dialer pool 1
 dialer idle-timeout 700000
 dialer called <own ISDN number 1>
 dialer called <own ISDN number 2>
 dialer max-call 1
 dialer-group 1
 x25 htc 1
 x25 win 7
 x25 wout 7
 no cdp enable

x25 route [X.25-ADDRESS-1] xot [IP-ADDRESS-RVS]
x25 route [X.25-ADDRESS-2] xot [IP-ADDRESS-RVS]
x25 route [X.25-ADDRESS-1] interface Dialer1

x25 host name [X.25-ADDRESS-ROUTER]
```

## Beispiel für CISCO Router Konfiguration (serial)

```
service pad to-xot
service pad from-xot
service tcp-keepalives-in
service tcp-keepalives-out
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname C2600
!
!
ip subnet-zero
isdn switch-type basic-net3
x25 routing
!
!
!
!
!
interface Ethernet0/0
 ip address 192.168.2.4 255.255.252.0
 no ip directed-broadcast
 no cdp enable
!
interface Serial0/0
 no ip address
 no ip directed-broadcast
 shutdown
!
ip classless
!
no cdp run
!
x25 route input-interface Serial0 xot 192.168.2.10
x25 route ^. interface Serial0/0
x25 host Router name1
!
line con 0
line aux 0
line vty 0 4
 login
!
end
```

## Beispiel CISCO Router 2800 für X.25

```
interface Serial0/0/0
 no ip address
 encapsulation x25
 no ip route-cache
 no ip mroute-cache
 x25 htc 15
 clock rate 2000000
 no cdp enable
!
x25 route ^. interface Serial0/0/0
```

## 7.3 BINTEC X4300-Konfiguration

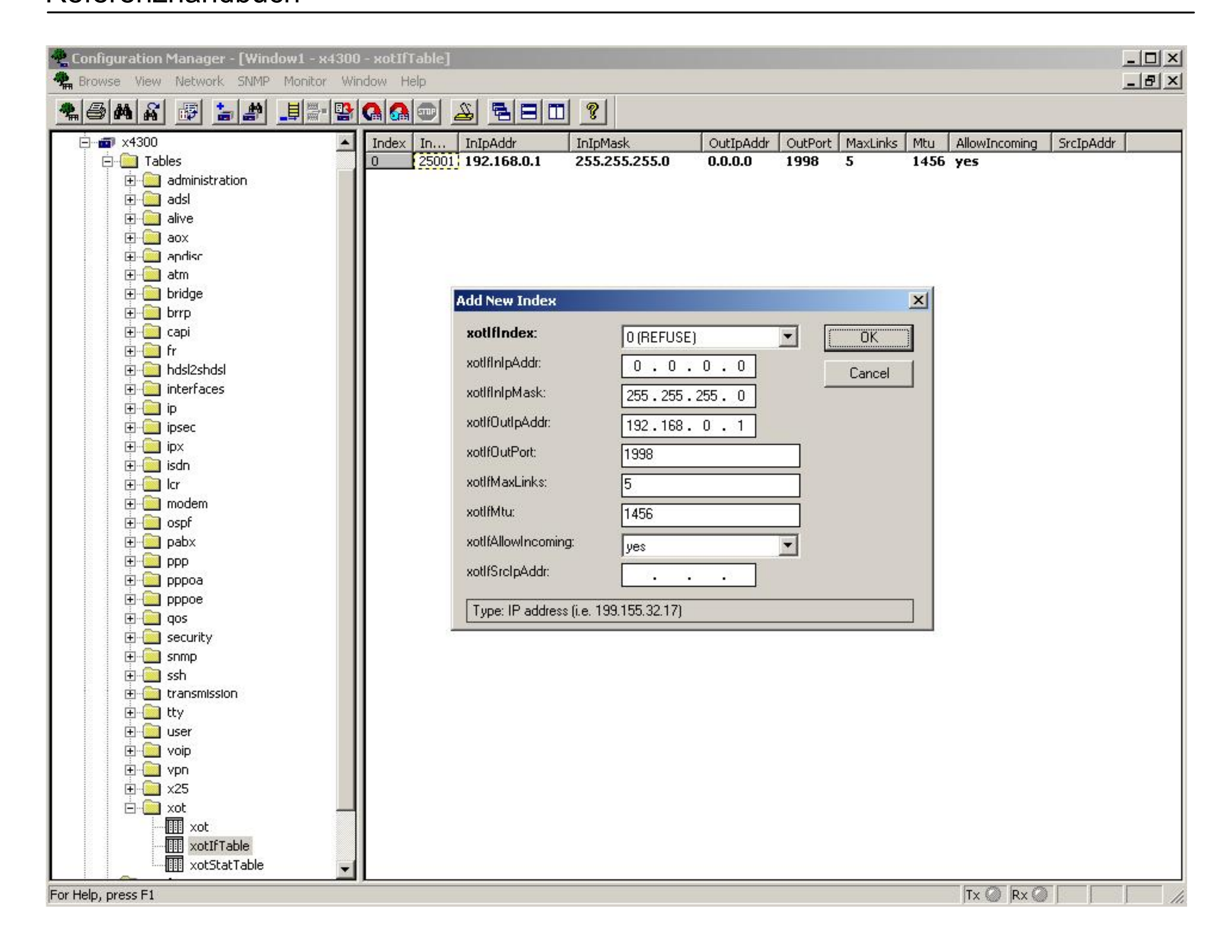
In diesem Kapitel bringen wir ein Beispiel für die Konfiguration des Routers BINTEC X4300.

In unseren Tests haben wir den BINTEC X4300 mit der firmware version 7.1 Rev. 1 eingesetzt.

Mehr Informationen über den BINTEC X4300 finden Sie hier:
http://www.funkwerk-ec.com/dl_bintec_x4x00_family_en,17190,837.html

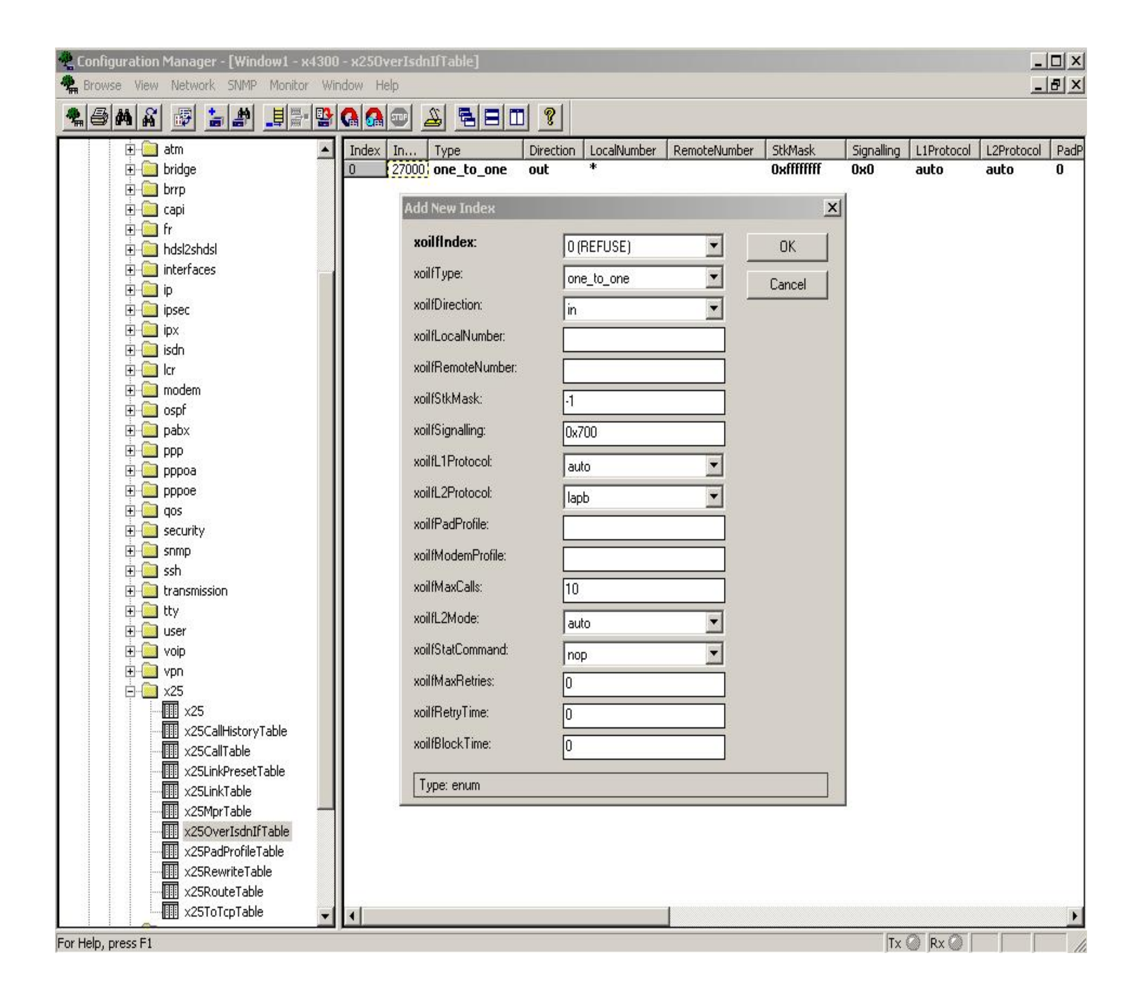
### 7.3.1 Konfiguration des BINTEC-Routers X4300 für XOT und XOI

1. Definiere die XOT-Schnittstelle (*xotIfTable*)
   - Sie müssen pro Partner und pro Richtung (eingehend/ausgehend) eine XOT-Schnittstelle definieren
   - Für eingehende XOT-Verbindungen muss der Parameter `xotIfOutIpAddr` auf 0.0.0.0 und der Parameter `xotIfInIpAddr` auf die IP-Adresse Ihrer Partnerstation gesetzt werden.
   - Für ausgehende XOT-Verbindungen müssen Sie den Parameter `xotIfInIpAddr` auf 0.0.0.0 und den Parameter `xotIfOutIpAddr` auf die IP-Adresse Ihrers Partners setzen.




2. Definiere die XOI-Schnittstelle (x25OverIsdnIfTable)

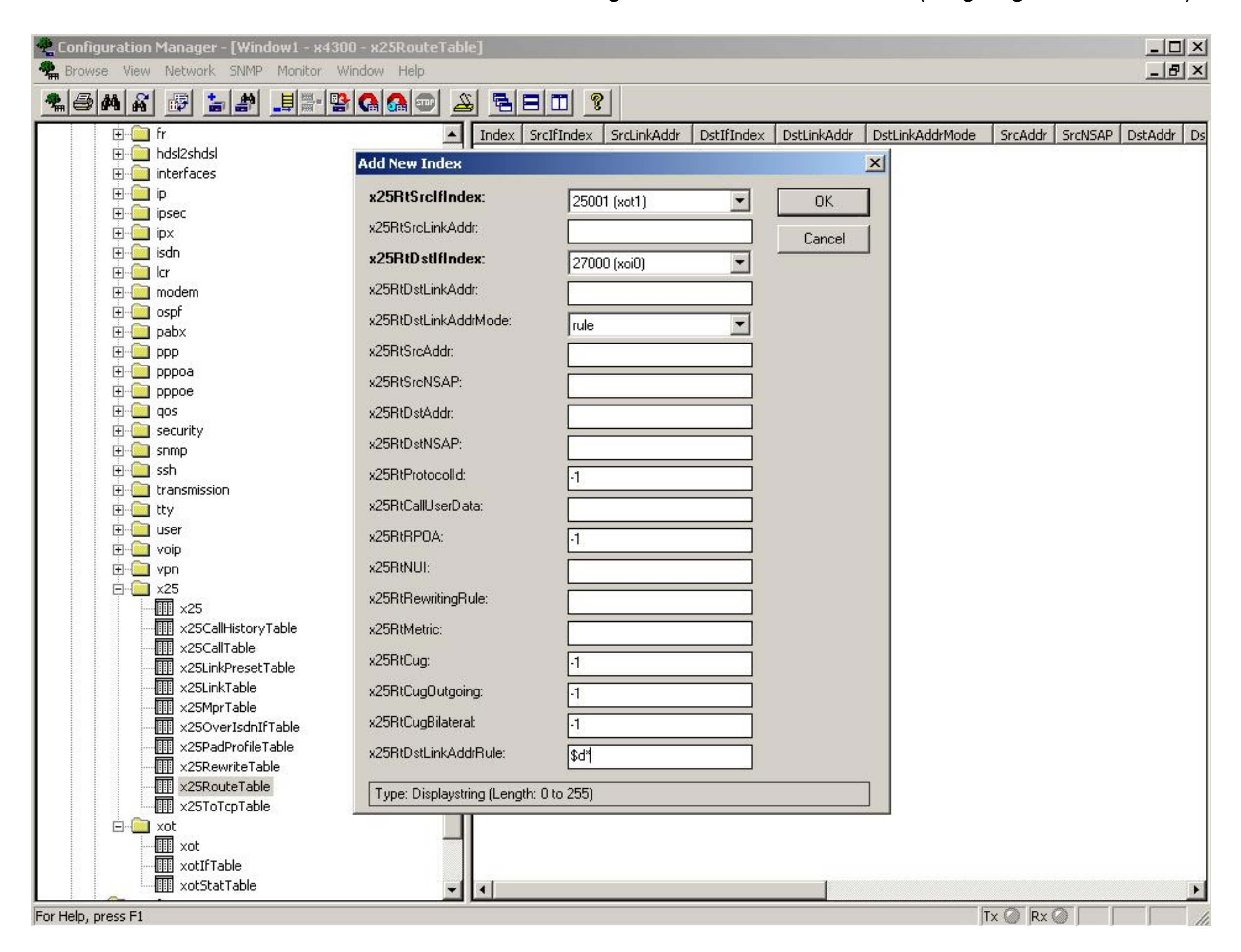
- Hier muss pro Kommunikationsrichtung eine Schnittstelle definiert werden.
- Für eine eingehende ISDN-Verbindung müssen Sie den Parameter `xoiIfDirection` auf `in`, den Parameter `xoiIfSignalling` auf `0x700` und den Parameter `xoiIfL2Protocoll` auf `lapb` setzen.
- Für eine ausgehende ISDN-Verbindung muss der Parameter `xoiIfRemoteNumber` auf `*` und der Parameter `xoiIfDirection` auf `out` gesetzt werden.
- Für die anderen Parameterwerte können die Standardeinstellungen belassen werden.




Nach diesen Konfigurationsschritten sollen Sie die Konfiguration speichern und den Konfigurationsmanager neu starten.

3. Define a route from an XOT interface to an XOI interface (x25RouteTable) Definiere eine Route von einer XOT-Schnittstelle zu einer XOI-Schnittstelle (x25RouteTable)

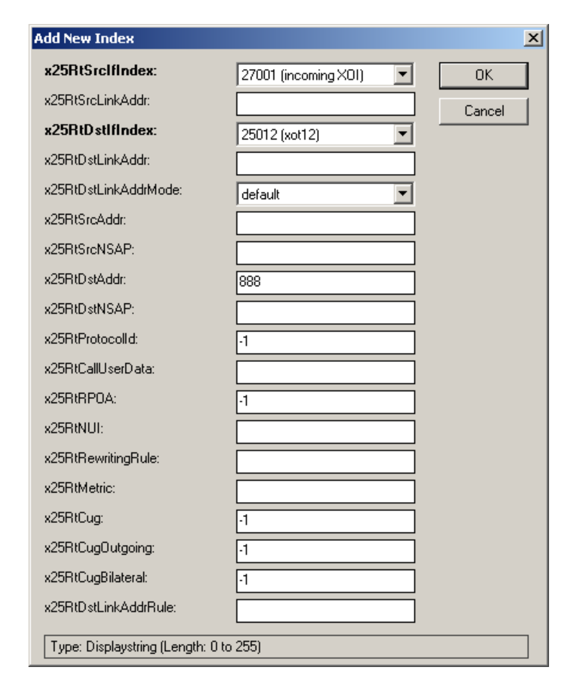
   - Pro Route muss man einen separaten Eintrag konfigurieren. Sie können auch eine Ersetzungsregel für die ISDN-Nummer verwenden (diese wird die Ziel-ISDN-Nummer durch die X.25-Adresse des X.25-Pakets ersetzen).
   - Setze den Parameter `x25RtDstLinkAddrRule` auf `$d*` (dies wird die ISDN-Nummer durch die X.25-Adresse ersetzen)
   - Setze den Parameter `X25RtDstLinkAddrMode` auf `rule` (dies wird die Ersetzungregel aktivieren).
   - Setze `x25RtSrcIfIndex` auf die eigehende XOT-Schnittstelle (incoming XOT interface) und `x25RtDstIfIndex` auf eine ausgehende XOI-Schnittstelle (outgoing XOI interface)

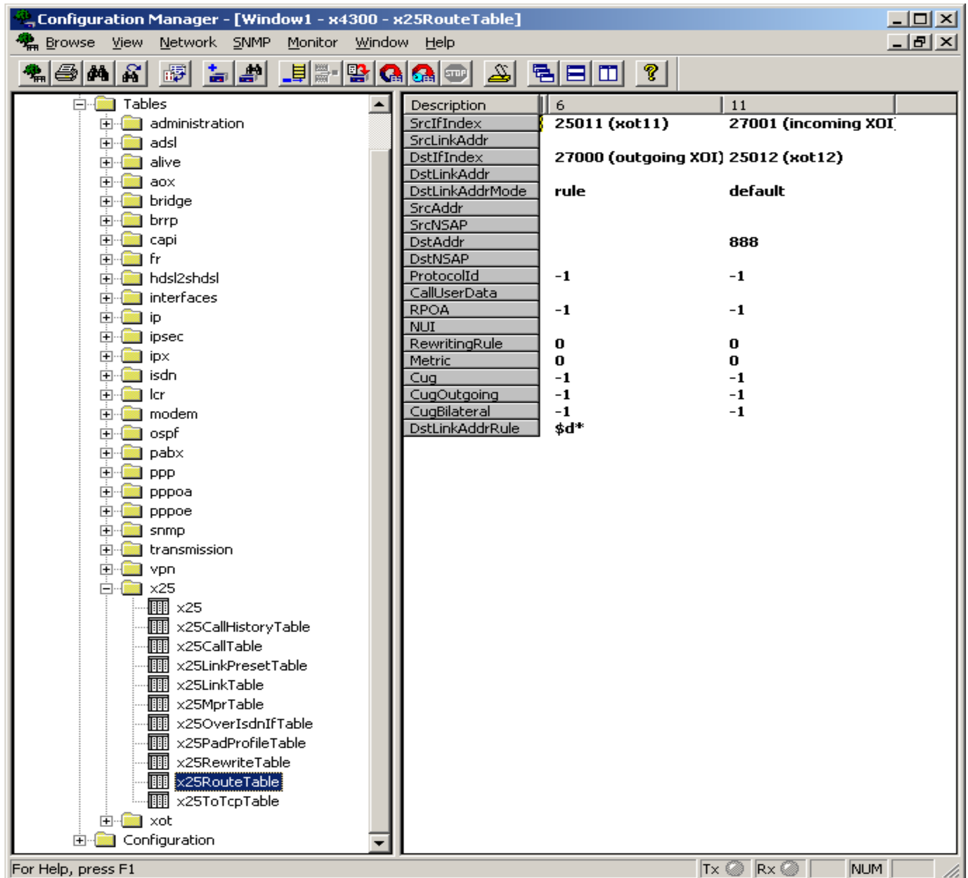



4. Definiere eine Route von einer XOI-Schnittstelle zu einer XOT-Schnittstelle

   - Pro Route muss man einen separaten Eintrag konfigurieren.
   - Für eine eingehende XOI-Verbindung muss die X.25-Adresse der Zielstation (Parameter `x25RtDstAddr`) angegeben werden. Der BINTEC-Router benutzt diese Adresse für das Routing der eingehenden X.25-Pakete zur richtigen Zielstation.
   - Sie müssen den Parameter `x25RtDstLinkAddrMode` auf seinem Standardwert belassen.

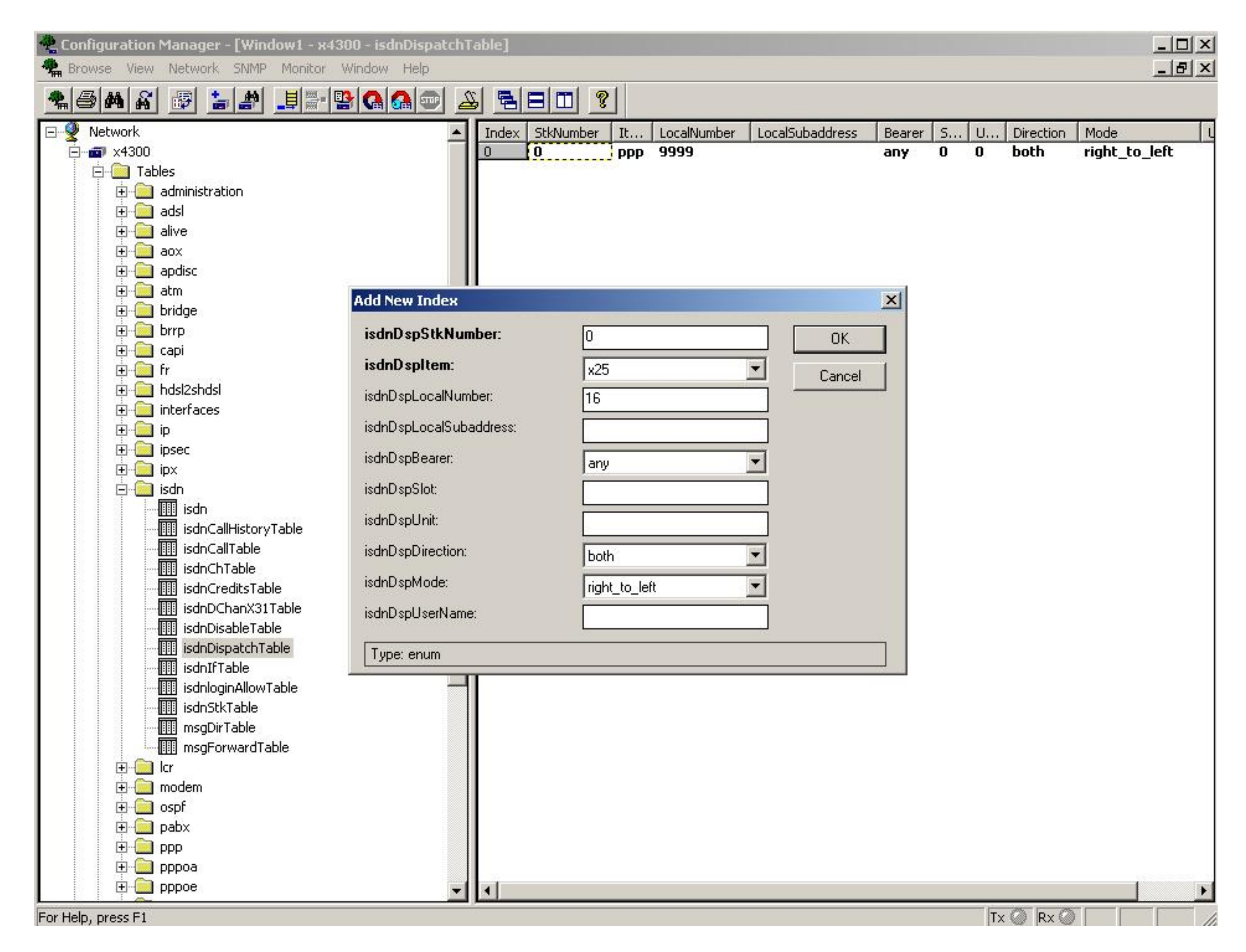
- `x25RtSrcIfIndex` muss auf die eingehende XOI-Schnitstelle und `x25RtSrcIfIndex` auf eine ausgehende XOT-Schnittstelle gesetzt werden.



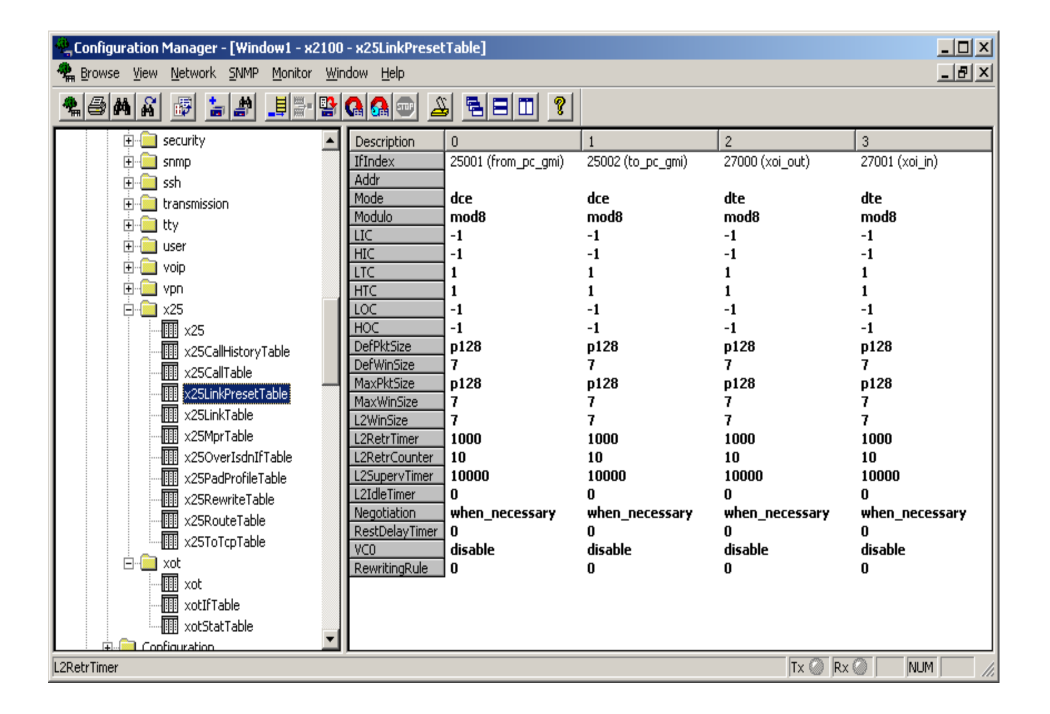
5. Konfiguriere die Antwort auf eingehende Rufe (Incoming call answering - IsdnDispatchTable)

- Setze den Parameter `isdnDspStkNumber` auf 0 und den Parameter `isdnDspItem` auf `x25` (Das wird den Anruf zur X.25-Schnittstelle weiterleiten.)
- Setze den Parameter `isdnDspLocalNumber` auf die lokale Nummer (MSN), die eingehende Anrufe verteilt.




6. Definiere für jede Schnittstelle X.25-Standardparameter (x25LinkPresetTable)

- Setze für jede XOI-Schnittstelle den Parameter `x25LkPrMode` auf DTE
- Setze für jede XOT-Schnittstelle den Parameter `x25LkPrMode` auf DCE
- Setze den Parameter `x25LkPrModulo` auf `mod8`
- Paket und Fenstergröße sollen den nationalen Gegebenheiten entsprechend angepasst werden.




## 7.3.2 Beispiel

Als Beispiel dient die Textdatei `BINTEC_X4300.txt` (unsere aktuelle Konfiguration). Diese Textdatei befindet sich auf unserer Distributions-CD, im Verzeichnis `_doc/ROUTER_CONF_EXAMPLES`.

# 8 Die rvs® Parameter

Der Betrieb des rvs® Monitors und der auf ihn bezogenen Komponenten kann von Änderungen der Parameterwerte beeinflusst werden. Im Folgenden wird beschrieben wie Sie auf verschiedenen rvs® portable-Plattformen die rvs®-Parameter konfigurieren oder anzeigen können.

**rvsX und rvs400**: Die Parameterwerte können entweder in der Datei `$RVSPATH/init/rdmini.dat` oder über die rvs® Konsole (`rvscns`) gesetzt werden. Mehr über die rvs®-Konsole lesen Sie bitte im rvsX Benutzerhandbuch, Kapitel "Operator-Konsole und Kommandos".

Die Festlegungen in der Datei `rdmini.dat` werden bei jedem neuen Start des rvs®-Monitors neu gelesen und sind daher sitzungsübergreifend gültig. Änderungen der Datei `rdmini.dat` werden erst durch Neustart des Monitors in die rvs®-Datenbank übertragen und dadurch aktiv. Befehle, die über die rvs®-Konsole gelesen werden, sind nur in der jeweiligen Sitzung gültig.

**rvsXP**: Zu den rvsXP-Parametern gelangen Sie aus dem rvsXP-Administrator, indem Sie die folgenden Schritte ausführen: `rvsXP-Administrator -> Ansicht -> Parameter`. Die Parameterwerte lassen sich ändern, indem Sie den zu konfigurierenden Parameter doppelklicken und in das Eingabefeld den gewünschten Wert eingeben. Diese Änderung wird erst nach Neustart des rvsXP-Monitors wirksam.

Alternativ können Sie einen Parameter auch im OperatorKommando-Dialog (`rvsXP-Administrator -> Bearbeiten -> OperatorKommando`) anzeigen oder verändern. Dies entspricht der rvs® Konsole (`rvscns`) in rvsX oder rvs400.

Folgende Syntax gilt dabei für alle rvs® portable (rvsX, rvsXP und rvs400):

**Syntax (Parameter verändern):**

`setparm PARM=VALUE` oder

`sp PARM=VALUE`

**Beispiel:** `sp ODTRACELVL=3`

In diesem Beispiel wird der rvs®-Parameter `ODTRACELVL` auf 3 erhöht, um die Trace-Dateien für die Fehleranalyse auf der ODETTE-Ebene ins `tracedir`-Verzeichnis von rvs® zu schreiben.

**Syntax (Parameter anzeigen):**

`listparm PARM` oder

`lp PARM`

Möglich ist auch:

`lp ALL`

um alle rvs®-Parameter anzuzeigen.

**Beispiel:** `lp TCPIPRCV`

**Hinweis:** Der Befehl `listparm` unterstützt Platzhalter. Als Platzhalter ist es möglich, einen Stern (*) zu verwenden. * entspricht einer unbestimmten Anzahl von beliebigen Zeichen.

Platzhalter müssen in einfachen oder doppelten Anführungszeichen angegeben werden.

**Beispiel**: `listparm "*prio"`

Dieser Befehl listet die Werte von **BBPRIO**, **IEPRIO**, **IZPRIO**, etc. auf.

### 8.1 Die rvs® Parameter im Überblick

| | |
|---|---|
| **ACTPCOUNT**<br>**(nur Parameter für aktives Panel)** | Angabe, nach wieviel Prozent einer Übertragung der Kommunikationsprozess wieder Informationen zum Prozess der Aktiven Leitungen (Active Panel) schickt.<br>Mögliche Werte: 1 - 100<br>Standard: **10** (Prozent) |
| **AECCHECK** | **A**uthority**E**xecute**C**ommand**C**heck. Überprüfe die Berechtigung, ein (rvs® Monitor-internes) Kommando auszuführen. Mögliche Werte: 0 (ausgeschaltet) und 1 (eingeschaltet).<br>Standard: **0** |
| **AGENT_HEARTBEAT** | Dieser Parameter ist nur im Zusammenhang mit dem rvs®-SNMP-Agenten von Bedeutung. Er definiert, ob eine Heartbeat-Meldung in regelmäßigen Abständen vom rvs®-Monitor an die TCP/IP-Adresse des rvs®-SNMP-Agenten gesendet wird.<br>Mögliche Werte: 0 (Nein) und 1 (Ja).<br>Standard: **0** |

**AGENT_LOGLEVEL** Dieser Parameter ist nur im Zusammenhang mit dem rvs®-SNMP-Agenten von Bedeutung. Er gibt an, welche Log-Meldungen an den rvs®-SNMP-Agenten gesendet werden sollen. Mögliche Werte: **0** = gar keine Log-Meldungen, **1** = alle, **2** = Fehler (Meldungsklasse: Error und Warning). Siehe bitte das Benutzerhandbuch rvs®-SNMP-Agent.

**AUTODECRYP** Bei AUTODECRYP=N wird eine empfangene Datei NICHT dekomprimiert und entschlüsselt, sondern so wie sie empfangen wurde, in das Verzeichnis `$RVS_PATH/usrdat` kopiert. Eine Dekomprimierung/ Entschlüsselung muss explizit vom Benutzer mit ComSecure gemacht werden.

Mögliche Werte: **N** (Nein) und **J** (Ja).
Standard: **J**

**BBCREATE** Generierung von Benutzerbenachrichtigungen (`BB` Kommando). Mögliche Werte: 0 (Nein) und 1 (Ja).

Standard: **0** (ausgeschaltet)

**BBPRIO** Priorität der Benutzerbenachrichtigungen (`BB` Kommando). Mögliche Werte: 1 – 100

Standard: **90**

**BRICKOFTPTI** Zeit in Sekunden, nach der das Warten auf Daten vom BRICK ISDN Adapter abgebrochen wird. Mögliche Werte: 20 – 600 Sekunden.

Standard: **20**

**CALLINGNUMCHECK** Zusätzlich zur Odette-ID wird die Rufnummer bzw. IP Adresse des Anrufers (Calling Party) geprüft.

Sofern die Rufnummer/IP-Adresse nicht dem erwarteten Wert entspricht, wird die Verbindung beendet.

Mögliche Werte: **N** (Nein) und **J** (Ja).
Standard: **N**

**CDWAIT** Die Zeitspanne in Sekunden vor einem OFTP Richtungswechsel (**C**hange **D**irection) nach Empfang einer Datei. Mögliche Werte: 0 – 5 Sekunden.

Standard: **1** (1 Sekunde)

**CHECKMAXPSESSIONS** Bei der Einstellung **CHECKMAXPSESSIONS = J** wird die Anzahl der gleichzeitigen Verbindungen auf den im Stationsparameter **PSESSION** angegebenen Wert begrenzt.

Mögliche Werte: **J** (Ja) und **N** (Nein)
Standard: **N**

**CMDDELETE**

Entferne jedes Kommando und die mit ihm verbundenen Einträge aus der rvs® Datenbank, sobald das Kommando ausgeführt oder gelöscht wurde. Mögliche Werte: 0 (ausgeschaltet) und 1 (eingeschaltet).

Standard: **1** (eingeschaltet)

**CNSMSGS**

IDs der Log-Nachrichten, die an die Operator-Konsole oder in der Log-Datei zu senden sind. Die folgenden Nachrichten-Codes sind definiert:

**A** Aktion

**B** Sicherheit

**E** Fehler

**I** Information

**L** Leitungstreiber

**O** Odette

**R** Report

**S** Gravierender Fehler

**W** Warnung

**+** lange Nachrichten

Standard: **BEILORSW+**

**CNTGC**

Dieser Parameter ist nur im Zusammenhang mit rvs® Data Center von Bedeutung; Anzahl der Kommandos, die ein Monitor abarbeiten muss, bevor er die rvs® Data Center-Überwachung durchführt. Siehe Benutzerhandbuch rvsX, Kapitel „rvs® Data Center“.

Standard: **40**.

**CNTIE**

Dieser Parameter steht in Zusammenhang mit dem Parameter **IECLEANTIME**. Nach jeweils **CNTIE** abgearbeiteten Kommandos wird geprüft, ob es in der Datenbank noch nicht abgearbeitete Einträge gibt, die älter sind, als mit **IECLEANTIME** eingestellt.
Standard: **1000**

**CNTMA**

Dieser Parameter ist nur im Zusammenhang mit rvs® Data Center von Bedeutung; Anzahl der Kommandos, die abgearbeitet werden müssen, bevor die Wiederherstellung von hängengebliebenen Aufträgen durchgeführt wird.

Standard: **40**

**CODEIN**

Eingabe-Code der lokalen Datei beim Erzeugen eines Sendeauftrags. Mögliche Werte **A** (für ASCII) und **E** (EBCDIC).

Standard: **A** (für ASCII)

**COMPFLAGS**

Die Version der offline-Komprimierung kann ab rvsX 4.06 konfiguriert werden. Mögliche Werte: `1` oder `2`. Für Dateien größer 4 GB ist Version `2` erforderlich, während ansonsten Version `1` oder `2` verwendet werden kann.

Standard: `2`

**Hinweis**: Die Parameter `COMPFLAGS` und `CRYPFLAGS` stehen im Zusammenhang. Wenn unterschiedliche Werte für `COMPFLAGS` und `CRYPFLAGS` angegeben werden, wird der kleinere Wert angewandt.

**COMTIMEOUT**

Dieser Parameter ist nur im Zusammenhang mit rvs® Data Center von Bedeutung; Timeout für den Sendeprozess (Sende-rvscom).

Standard: **700** (Sekunden).

**CRYPFLAGS**

Die Version der Verschlüsselung kann ab rvsX 4.06 konfiguriert werden. Mögliche Werte: 1 oder 2. Für Dateien größer 4 GB ist Version 2 erforderlich, während ansonsten Version 1 oder 2 verwendet werden kann.

Standard: 2

**Hinweis**: Die Parameter `COMPFLAGS` und `CRYPFLAGS` stehen im Zusammenhang. Wenn unterschiedliche Werte für `COMPFLAGS` und `CRYPFLAGS` angegeben werden, wird der kleinere Wert angewandt.

**DBLOGMAXENTRIES**

Bei einem rvs® mit einer externen Datenbank ist es möglich, die Log-Meldungen in die Datenbank zu schreiben (Parameter `LOGINDB` in der rvs®-Umgebungsdatei `rvsenv.dat`, siehe Benutzerhandbuch). Diese Log-Meldungen können auch automatisch auf Platte gesichert werden, damit die Datenbanktabelle LT, in der die Meldungen gespeichert werden, nicht überläuft. Mit dem Parameter **DBLOGMAXENTRIES** wird die maximale Anzahl von Log-Meldungen in der Datenbank angegeben. Wird dieser Wert überschritten, wird der ältere Teil der Datenbank-Tabelle LT in eine Datei ($RVSPATH/arcdir/dblogmessages.<timestamp>) exportiert und in der Datenbank gelöscht.

Standard: 50000 Meldungen

| | |
|---|---|
| **DBLOGMINENTRIES** | Wenn Log-Meldungen aus der Datenbank (LT-Tabelle) exportiert werden (Siehe Parameter **DBLOGMAXENTRIES** in dieser Tabelle), entscheidet der Parameter **DBLOGMINENTRIES**, wie viele Log-Meldungen zurückbehalten werden. Es werden immer die neuesten Meldungen zurückbehalten.<br><br>Standard: 10000 |
| **DIALCOUNT** | Nur für ISDN/X.25. Anzahl der Wahlversuche bei einer besetzten Leitung („busy“). Wenn diese nicht erfolgreich sind, versucht rvs® mit einer anderen Leitung (siehe Kapitel über alternative Netzwerke im Benutzerhandbuch) die Verbindung aufzubauen. Siehe auch Parameter **DIALRTIME** in dieser Tabelle.<br><br>Standard: 5 Versuche |
| **DIALRTIME** | Zeit in Sekunden, die gewartet wird, bevor ein neuer Versuch auf einer ISDN/X.25-Leitung im Status „busy (besetzt)“ unternommen wird. Dieser Parameter steht in engem Zusammenhang mit dem Parameter **DIALCOUNT** (Anzahl der Wahlversuche, siehe diese Tabelle).<br><br>Standard: 5 (Sekunden) |
| **DISKSPACEREJECT** | Anzahl von Kilobytes, die imVerzeichnis `$RVSPATH/temp` frei sein müssen, damit Dateien empfangen werden können.<br><br>Standard: 5000 (ca. 5 MByte). Siehe Kapitel 8.3 für mehr Informationen. |
| **DISKSPACEWARN** | Anzahl von Kilobytes in jedem auf Speicherplatzmangel zu überprüfenden rvs®-Verzeichnis, die frei sein muss. Ein Unterschreiten bewirkt Aufruf des Skripts `diskspacewarn`.<br><br>Standard: 5000 (ca. 5 Mbyte). Siehe Kapitel 8.3. für mehr Informationen. |
| **DTCONNnn** | Zeitabschnitte, die abzuwarten sind, bevor ein Verbindungsversuch wiederholt wird.<br><br>**nn** ist die Anzahl der erfolglosen Verbindungsversuche (**CNTRETRY** in `SK`). Sie brauchen keinen Parameter für jeden Wert von **nn**; wenn einer davon undefiniert bleibt, wird der nächst kleinere, der sich findet, verwendet.<br><br>Format: MM/DD/YY HH:MM:SS<br><br>Standards: Größere Zeitabschnitte, damit rvs® nicht ständig mit Versuchen überlastet wird, eine Station zu erreichen, die vielleicht Hardwareprobleme hat. Für |

| | |
|---|---|
| | längere Zeitspannen werden zum Normalfall weitere Minuten addiert, so dass keine Wiederholungsversuche gleichzeitig stattfinden:<br>**DTCONN01** "00/00/00 00:01:00"<br>**DTCONN02** "00/00/00 00:02:00"<br>**DTCONN03** "00/00/00 00:03:00"<br>**DTCONN05** "00/00/00 00:05:00"<br>**DTCONN07** "00/00/00 00:07:00"<br>**DTCONN10** "00/00/00 00:10:00"<br>**DTCONN15** "00/00/00 00:15:00"<br>**DTCONN20** "00/00/00 00:20:00"<br>Zusätzlich gibt **DTCONN01** die Wartezeit für alle anderen rvs®-Kommandos an. |
| **DTCOPY** | Wenn es einen Fehler bei der internen Zustellung einer Datei gibt - d.h. beim Kopieren vom `$RVS_PATH/temp` Verzeichnis in das Verzeichnis `$RVS_PATH/usrdat`, wird nach Ablauf dieses Intervalls die interne Zustellung eiederholt.<br>Zeitintervall im Format YY/MM/DD hh:mm:ss, Standard: 00/00/00 00:10:00 d.h. 10 Minuten |
| **EERP_IN** | Bestätigung (EERP=End-to-End-Response) bei der Sendeübertragung. Mögliche Werte:<br>**NEVER**: Partner sendet keinen EERP, das Versenden einer Datei endet mit erfolgreicher Übertragung. Kein Warten auf Quittung.<br>**NORMAL**: Warte nach erfolgreicher Übertragung einer Datei auf Quittung durch Partner. Versenden einer Datei endet mit Erhalt des EERPs.<br>Standard: **NORMAL** |

**EERP_OUT**

Handhabung für das Senden von Empfangsbestätigungen (EERPs).

**NEVER** Partner akzeptiert keine Empfangsbestätigung; es werden keine EERPs generiert. Der Empfang einer Datei endet mit erfolgreicher Übertragung.

**IMMEDIATE** Generieren einer Empfangsbestätigung nach erfolgreichem Empfang einer Datei und unmittelbares, aktives Versenden.

**NORMAL** Nach erfolgreichem Empfang einer Datei Empfangsbestätigung erstellen, EERP nur bei noch bestehender Verbindung an den Partner senden, sonst bei der nächsten Verbindung EERP senden.

**SYNC** die Empfangsbestätigung wird in der gleichen Sitzung in der Datei empfangen wurde, versandt. Die Verbindung wird dazu bei Bedarf so lange aufrechterhalten, bis der EERP (nach erfolgreicher Zustellung) generiert ist. Mit dem rvs® -Parameter **SYNCDL** können Sie Zeit in Millisekunden (ms) angeben, die abgewartet werden soll, bis geprüft wird, ob der EERP zum Versand bereit steht. Die Anzahl der Wartevorgänge wird mit dem rvs® Parameter **SYNCO** konfiguriert. Ist beispielsweise **SYNCDL**=400 und **SYNCTO**=5, wird maximal 5-mal 400 ms gewartet, bis die Verbindung geschlossen wird. Wenn der EERP in dieser Zeit zum Versand bereit steht, wird er übertragen und die Verbindung danach beendet.

**HOLD** Generieren einer Empfangsbestätigung nach erfolgreichem Empfang einer Datei. Empfangsbestätigung wird jedoch erst nach Freigabe ein EERP nur bei noch bestehender Verbindung an den Partner senden, sonst bei der nächsten Verbindung EERP senden.

**HOLDIMMED**: Nach erfolgreichem Empfang einer Datei Empfangsquittung erstellen. Quittung jedoch nicht versenden, sondern für Freigabe durch Personal festhalten. Nach der Freigabe: Wenn keine Verbindung zum Partner besteht, Verbindung herstellen und EERP an den Partner senden. Quittung wieder freigeben: (siehe Benutzerhandbuch, OP-Parameter)

Standard: **NORMAL**

**Hinweis:** Standard-Einstellung für alle Stationen (Zeile OP-Odette Parameter) ist: `EERP_OUT=IMMEDIATE` (siehe Benutzerhandbuch, OP-Parameter)

**EFIDGAPTIMEOUT** Bei einem unklaren Status einer Übertragung wird das mit **EFIDGAPTIMEOUT** eingestellte Intervall gewartet, bevor die Übertragung erneut durchgeführt wird.

Zeitintervall im Format YY/MM/DD hh:mm:ss,
Standard: 00/00/00 01:00:00 d.h. 1 Stunde

**FORCEDEND** Anhalten des rvs® Monitors: sofortiges Beenden gefordert, selbst wenn der Transmitter und der Empfänger aktiv sind.

**Hinweis**: Wenn der Parameter auf "1" gesetzt ist, wird der Monitor sofort gestoppt. Mögliche Werte: 0 (ausgeschaltet) und 1 (eingeschaltet).

Standard: **0** (aus)

**HEAVYDUTY** Eine Überlastungsanzeige. Es wird immer dann ein Eintrag in die Log-Datei geschrieben, wenn das rvs System mit der Bearbeitung der aktuell anstehenden Aufgaben voraussichtlich länger beschäftigt sein wird, als mit **HEAVYDUTY** eingestellt.Gilt nur bei ISAM-Systemen.

Zeitintervall im Format YY/MM/DD hh:mm:ss,
Standard: 00/00/00 01:00:00 d.h. 1 Stunde

**IECLEANTIME** IE (Informationseinträge) zu nicht abgearbeiteten oder unterbrochenen Kommandos, die bereits länger in der Datenbank gehalten werden als mit **IECLEANTIME** definiert, werden gelöscht. Steht in Zusammenhang mit **CNTIE**.
Zeitintervall, Format YY/MM/DD hh:mm:ss,
Standard: 00/00/14 00:00:00 d.h. 14 Tage

**IEPRIO** Priorität der InformationsEingangs (`IE)` Kommandos. Mögliche Werte: 1 - 100.

Standard: **50**

**INITCMDS** Führe Initialisierungskommandos aus der Datei `$RVS_PATH/init/rdmini.dat`. bei Neustart des rvs®-Monitors. Mögliche Werte: 0 (Nein) und 1 (Ja).

Standard: **1** (eingeschaltet).

**IZPRIO** Priorität der InformationsZustellung-Kommandos (`IZ)`. Mögliche Werte: 1 – 100.

Standard: **40**

**JSERRHOLD** Entscheidet, ob ein Sendeauftrag, der fehlgeschlagen ist und für den ein Jobstart nach Sendeversuch gestartet wird, angehalten wird oder nicht.

Falls der Parameter den Wert **Ja** oder **Yes** hat, wird der

| | |
|---|---|
| | Sendeauftrag angehalten (in den Status ‚hold' gesetzt).<br>Falls der Parameter den Wert **Nein** oder **No** hat, versucht rvs®, den Sendeauftrag weiter auszuführen.<br>Standard: **Nein** |
| **KEEPDAYS** | Anzahl der Tage, nach der gelöschte und ausgeführte Kommandos und die mit ihnen verbundenen Informationen, während der Datenbankreinigung aus der rvs® Datenbank endgültig entfernt werden.<br>Standard: **7** |
| **LANGUAGE** | Sprache der Bedienerkommunikation und der Log-Nachrichten:<br>**D** Deutsch<br>**E** Englisch<br>Standard: **E** |
| **LDSNPRIO** | **L**ong**D**ata**S**et**N**umber**P**rority. Sendepriorität für große Dateien. Mögliche Werte: 1 – 100 (ganzzahlig). Je kleiner der Wert, desto größer die Priorität. Dieser Parameter soll im Zusammenhang mit den Parametern **SDSNPRIO** und **SDSNMAX** verwendet werden. Wenn **LDSNPRIO** größer ist als **SDSNPRIO** (Standard-Einstellung) werden kleinere Dateien vorrangig versendet.<br>Standard: **50** |
| **LID** | Lokale Stations-ID<br>Standard: geliefert während der rvs® Datenbank-Initialisierung |
| **LITRACELVL** | Anforderung einer Leitungsverfolgung (zwischen OFTP und dem Netzwerk):<br>0. Keine Verfolgung<br>1. Minimale Verfolgung (Leitungstreiber-Ereignisse usw. ) für die im Parameter **SIDTRACE** spezifizierte Station<br>2. Ausführliche Verfolgung (inkl. Hex Dump der Daten) für die im Parameter **SIDTRACE** spezifizierte Station<br>3. Ausführliche Verfolgung für alle Stationen<br>Standard: **0** |
| **LMPRIO** | Priorität für Log-Nachrichten als externe LM Kommandos. Mögliche Werte: 1 – 100.<br>Standard: **20** |

| | |
|---|---|
| **MAXCMD** | Maximale Anzahl gleichzeitig gelesener Kommandos. Mögliche Werte: 1 – n.<br>Standard: **10** |
| **MAXRECL** | Maximale Record-Länge für Dateien im Format **F** oder **V** beim Empfang.<br>Mögliche Werte: 1 – n. Standard: **99999** |
| **MAXSENDERS** | Maximale Anzahl von aktiven Sendern. Wenn **MAXSENDERS**=**0,** wird kein Sender gestartet. Mögliche Werte: 0 – n.<br>Standard: **1** |
| **MAXX25RCV** | Maximale Anzahl der gleichzeitig aktiven oder voraktivierten `horchenden` (listening) Prozesse für X.25/ISDN Kommunikation. Mögliche Werte: 0 – n.<br>Standard: **0 (nicht aktiv)** |
| **MONTIMEOUT** | Dieser Parameter ist nur im Zusammenhang mit rvs® Data Center von Bedeutung; Timeout für rvs®-Monitore.<br>Standard: **700** (Sekunden). |
| **MSGPRIO** | Sendepriorität für Nachrichten zwischen Operatoren. Mögliche Werte: 0 – 100.<br>Standard: **60** |
| **MWSTART** | Gleichzeitiges Starten von rvs® und rvs® Middleware. Mögliche Werte: 0 (kein gleichzeitiges Starten); 1 (gleichzeitiges Starten) |
| **MWSTOP** | Gleichzeitiges Stoppen von rvs® und rvs® Middleware. Mögliche Werte: 0 (kein gleichzeitiges Stoppen); 1 (gleichzeitiges Stoppen) |
| **MWTIMEOUT** | Die Aktivitäten der rvs® Middleware werden in die rvs®-Datenbanktabelle RI geschrieben. Dieser Parameter gibt an, nach wie viel Zeit ohne Aktualisierung der RI-Tabelle die Prozesse der rvs®-Middleware beendet werden. |
| **NUMREDOLOGS** | Anzahl der `redo.log`-Dateien (Funktionalität: Backup/Recovery, siehe Benutzerhandbuch), die generiert werden können.<br>Mögliche Werte: 1 – NOLIMIT.<br>Standard: **NOLIMIT**<br>Hinweis: Wenn `NUMREDOLOGS` den Wert `NOLIMIT` hat, erhalten unterschiedliche Log-Dateien einen Zeitstempel, der die Anzahl der vergangenen Sekunden ab 1.1.1970 zählt (Unix-Zeit). Bei anderen numerischen |

| | |
|---|---|
| | Werten, wird der Name der Datei, um den numerischen Wert erweitert (z.B. redo.001).<br>Standard: **NOLIMIT** |
| **NUMRLOGS** | Anzahl der `rlog.log` Log-Dateien, die generiert werden können. Mögliche Werte: 1 – NOLIMIT.<br>Standard: **NOLIMIT**<br>Hinweis: Wenn `NUMRLOGS` den Wert `NOLIMIT` hat, erhalten unterschiedliche Log-Dateien einen Zeitstempel, der die Anzahl der vergangenen Sekunden ab 1.1.1970 zählt (Unix-Zeit). |
| **NUMRLSTAT** | Anzahl der `rlstat.log` Log-Dateien, die generiert werden können. Mögliche Werte: 1 – NOLIMIT.<br>Standard: **NOLIMIT**<br>Hinweis: Wenn `NUMRLSTAT` den Wert `NOLIMIT` hat, erhalten unterschiedliche Log-Dateien einen Zeitstempel, der die Anzahl der vergangenen Sekunden ab 1.1.1970 zählt (Unix-Zeit) |
| **OCREVAL** | Odette Kreditwert (credit value) = Fenstergröße von OFTP:<br>Maximale Anzahl von Sendeblöcken ohne Erwartung einer Quittung. Setzen Sie den Wert umso höher, je besser die Leitungsqualität ist, um eine gute Übertragungsleistung zu erreichen. Mögliche Werte: 1..n<br>Standard: **99** |
| **ODTRACELVL** | Fordere eine Leitungsverfolgung an (zwischen dem Sender und dem OFTP):<br>0 Keine Verfolgung<br>1 Minimale Verfolgung (nur Namen) für die in **SIDTRACE** spezifizierte Station.<br>2 Ausführliche Verfolgung (Parameterwerte etc.) für die in **SIDTRACE** spezifizierte Station.<br>3 Ausführliche Verfolgung für alle Stationen.<br>Standard: **0** |
| **OEXBUF** | Größe des Odette-Austauschpuffers in Bytes (das größte Odette-Kommando (**SFID**)); Setzen Sie den Wert auf ein Vielfaches von 128 und möglichst hoch bei guter Leitungsqualität, um eine gute Übertragungsleistung zu erreichnen.<br>Mögliche Werte: 128…64 kB.<br>Standard: **2048** |

**OKPRIO** Priorität für Operator-Kommandos. Mögliche Werte: 1 – 100.

Standard: **10**

**ORETRY** zeigt die Odette-Fehlergruppe an, für die ein Wiederholungsversuch gestartet wird, nachdem die Anfrage unterbrochen worden ist. Angabe erfolgt in Form eines Bitfelds, wobei die Bitposition dem Fehlercode zugeordnet ist. 1 bedeutet: Wiederaufsetzen wird durchgeführt, 0 dagegen: Wiederaufsetzen wird unterdrückt.

Die Bits entsprechen folgenden Fehlern:

- 1 – Übertragung wurde unterbrochen
- 2 – Datei wurde nicht gefunden oder konnte nicht geöffnet werden
- 3 – Datei konnte nicht gelesen werden
- 4 – Fehlercode „File size is too big“ im SFNA mit Wiederholung erlaubt
- 5 – Fehlercode „ Unspecifed reason“ im SFNA mit Wiederholung erlaubt
- 6 – Fehlercode „File size is too big“ im SFNA mit Wiederholung nicht erlaubt
- 7 - Fehlercode „ Unspecifed reason“ im SFNA mit Wiederholung nicht erlaubt
- 8 - Fehlercode „File size is too big“ im EFNA
- 9 – Fehlercode „Invalid record count“ im EFNA
- 10 - Fehlercode „Invalid byte count“ im EFNA
- 11 – Fehlercode „Access method failure“ im EFNA
- 12 – Fehlercode „„ Unspecifed reason“ im EFNA

Standard: **10111000111011111111**

**OTIMEOUT** Odette Time-Out: Abbruchzeit in Sekunden, nach der der Sender die Verbindungsaufnahme abbricht, wenn die Partnerstation oder das CAPI nicht antworten.

Standard: **600** (Sekunden)

**QEPRIO** Priorität des Quittungsempfangs (`QE`-Kommandos). Mögliche Werte: 1 – 100.

Standard: **30**

**QSPRIO** Priorität für das Senden von Quittungen (`QS`-Kommandos). Der Wert sollte zwischen **MSGPRIO** und

| | |
|---|---|
| | **SDSNPRIO** liegen. Mögliche Werte: 1 – 100.<br>Standard: **30** |
| **RECVBLOCKS** | Zahl der empfangenen Blöcke zwischen zwei Wiederaufsetzpositionen für die Fehlerkorrektur = Größe des temporären Datenpuffers. Setzen Sie den Wert umso höher, je besser die Leitungsqualität ist, um eine gute Übertragungsleistung zu erreichen.<br>Standard: **1000** |
| **REDOMAXSIZE** | Maximale Größe in Bytes für die Datei: Datei `$RVS_PATH/arcdir/redo.log`<br>Standard: **1000000** |
| **RLCOMAXSIZE** | Maximale Größe in Bytes für die Datei: Datei `$RVS_PATH/db/rlco.log` (Konsolennachrichten)<br>Standard: **100000** |
| **RLDB** | Loggen von Datenbank-Aktionen in die Datei `$RVSPATH/db/rldb.log`. Mögliche Werte: **0** (ausgeschaltet);<br>**1** (eingeschaltet). Standard: **0**. |
| **RLDBMAXSIZE** | Maximale Dateigröße in Bytes für das Loggen von Datenbank-Aktionen in die Datei `$RVSPATH/db/rldb.log`<br>Standard: **1000000** (1 Mbyte), keine Begrenzung |
| **RLOGMAXSIZE** | Maximale Größe in Bytes für die Datei `$RVSPATH/db/rlog.log` (Log-Nachrichten)<br>Standard: **2000000** (2 Mbyte), keine Begrenzung<br>**Hinweis**: Dieser Parameter steht im Zusammenhang mit dem Parameter `NUMRLOGS`.<br>Wenn `NUMRLOGS = NOLIMIT` erhalten unterschiedliche Log-Dateien einen Zeitstempel, der die Anzahl der vergangenen Sekunden ab 1.1.1970 zählt (Unix-Zeit). |
| **ROUTING** | Legt fest, ob über die lokale Station „Routing“ möglich ist. In manchen Anwendungsszenarien ist es vorteilhaft, das OFTP-Routing nicht zuzulassen. Dies ist rvs®-global mit dem rvs® Parameter **ROUTING** in der Datei `rdmini.dat` oder für einzelne Stationen in der OP-Konfiguration (siehe Benutzerhandbuch) mit dem gleichlautenden Eintrag möglich.<br>**I** (IN): Eingehende Dateiübertragung vom Partner z.B. `XXX` zum remote-Partner `REM1` über Ihre lokale Station<br>`LOC` ist erlaubt (`XXX` ⇒`LOC` ⇒`REM1`); nicht erlaubt ist ausgehendes Routing z.B. für den Partner `REM2` über |

| | |
|---|---|
| | REM1 (LOC ⇒REM1⇒ REM2). |
| | **O** (OUT): Partnerstationen können Ihre lokale Station nicht als Router benutzen. Erlaubt ist: ausgehendes Routing z.B. für den Partner REM2 über Partner REM1. Nicht erlaubt ist: Ausgehende Dateiübertragung vom Partner z.B. XXX zum entfernten Partner REM1 über Ihre lokale Station LOC (XXX ⇒LOC ⇒REM1). |
| | **B** (BOTH): bedeutet normales OFTP-Routing |
| | **N** (NEVER): Routing in beide Richtungen (IN und OUT) ist verboten. |
| | Default: **B** |
| **RSTATMAXSIZE** | Maximale Größe in Bytes für die Datei $RVSPATH/db/rlstat.log (statistische Logs) |
| | Standard: **2000000** (2 Mbyte), keine Begrenzung |
| | **Hinweis**: Dieser Parameter steht im Zusammenhang mit dem Parameter NUMRLOGS.<br>Wenn NUMRLOGS = NOLIMIT erhalten unterschiedliche Log-Dateien einen Zeitstempel, der die Anzahl der vergangenen Sekunden ab 1.1.1970 zählt (Unix-Zeit). |
| **RVSDIAEXTENDED-MODE** | Erweiterte Anzeige der Informationen zu den gerouteten Empfangsaufträgen in rvsdia. Die erweiterte Anzeige wird ausgegeben, wenn **RVSDIAEXTENDEDMODE=1**, **J**, **j**, **Y** oder **y** ist. Standardwert ist **0** (Anzeige ohne Erweiterung); |
| **SCPRIO** | Häufigkeit der ServiceProvider-Überprüfung durch den rvs®-Monitor im Verzeichnis SPOUTDIR. Siehe Datei $RVSPATH/rvsenv.dat für den Wert von SPOUTDIR. |
| **SDSNMAX** | Maximale Größe einer Datei, um als klein eingestuft zu werden (in Einheiten 1024 Byte) |
| | Standard: **100** (bedeutet: 100 mal 1024 Byte) |
| **SDSNPRIO** | **S**hort**D**ata**S**et**N**umber**PRIO**rity |
| | Sendepriorität für kleine Dateien. Mögliche Werte 1 -100 (ganzzahlig). Je kleiner der Wert, desto größer die Priorität. Dieser Parameter soll im Zusammenhang mit den Parametern **LDSNPRIO** und **SDSNMAX** verwendet werden. Wenn **LDSNPRIO** größer ist als **SDSNPRIO** (Standard-Einstellung), werden kleinere Dateien vorrangig versendet. Mögliche Werte: 1 – 100. |
| | Standard: **40** |
| **SECURITY** | Dieser Parameter regelt wie die Datenverschlüsselung |

beim Dateitransfer angewandt wird. Der kann für alle Stationen global in der Datei `$RVSPATH/init/rdmini.dat` oder stationsweise in der OP-Konfiguration gesetzt werden. Der globale Monitorparameter aus der Datei `rdmini.dat` wird für Stationen herangezogen, für die der OP-Parameter nicht gesetzt ist.

**NO**: Verschlüsselung nicht möglich. Falls ein Sendejob Verschlüsselung verlangt, wird der Job mit einer Fehlermeldung abgebrochen.

**OPT**: Verschlüsselung optional möglich und kann im Sendejob festgelegt werden.

**FORCED**: Verschlüsselung zwingend. Falls ein Sendejob die Verschlüsselung nicht vorsieht, wird eine Warnung ausgegeben und der Sendejob in einen Job mit Verschlüsselung umgewandelt. Falls die Partnerstation eine Datei unverschlüsselt sendet, wird der Empfang der Datei abgelehnt. Ein Sendejob für die Station 'S' wird gemäß dem **SECURITY** Eintrag für Station 'S' behandelt, unabhängig davon, ob 'S' eine Nachbarstation ist oder über Routing erreicht wird.

Mehr über Verschlüsselung lesen Sie bitte im Benutzerhanbuch.

Default: **SECURITY=OPT**

**SENDBLOCKS**

Zahl der gesendeten Blöcke zwischen zwei Status-Prüfungen des Parameters **FORCEDEND**. Setzen Sie den Wert umso höher, je besser die Leitungsqualität ist, um eine gute Übertragungsleistung zu erreichen.

Standard: **1000**

**SEPRIO**

Die Priorität für neue `SE`s sollte mindestens so hoch sein wie die höchste Priorität gültig für `SE`.

Standard: **50**

**SIDTRACE**

ID der Station, die verfolgt werden soll (wenn **LITRACELVL** oder **ODTRACELVL** mindestens auf **1** oder **2** gesetzt sind).

Standard is "   " (3 Leerzeichen).

Wenn Sie eingehende Daten tracen wollen, müssen Sie **SIDTRACE** gleich der lokalen Stations-ID (**LID**) setzen.

**SLEEP**

Die Zeitspanne in Sekunden, die der rvs® Monitor abwartet, wenn es nichts Weiteres zu tun gibt, bevor er überprüft, ob ein neues Kommando zur Ausführung wartet.

Standard: **30** (Sekunden)

**SNARCV** Ermöglicht den automatischen Start des SNA Transaktionsprogramms, wenn Anrufe eingehen:

0 das Transaktionsprogramm wird nicht gestartet

1 das Transaktionsprogramm wird gestartet

Standard: **0**

**SSCREATE** Generieren einer Sendestatistik-Aufzeichnung für jeden Übertragungsversuch

Standard: **0** (ausgeschaltet)

**STATCHECKINT** Nach Ablauf von **STATCHECKINT** erfolgt eine Aufnahme des Systemstatus. Zeitintervall, Format YY/MM/DD hh:mm:ss,
Standard: 00/00/00 01:00:00 d.h. 1 Stunde

**STATISTICS** Generieren einer Sendestatistik-Aufzeichnung in der Statistik-Log-Datei (`rlstat.log` für UNIX und Windows NT)

0 keine Statistik-Log-Datei

1 kurze Form

2 ausführliche Form der Statistik

3 kurze Form der Statistik mit gerouteten Übertragungen

4 ausführliche Form der Statistik inkl. gerouteter Übertragungen

5 neue Parameter wie Dateiformat, Zustand der Übertragung, Anzahl der Einwahlversuche

6 Statistik über gelöschte Einträge (auch vom Benutzer)

7 Statistik über gelöschte Einträge und über geroutete Übertragungen

Standard: **2** (ausführliche Statistik eingeschaltet)

**SYNCDL** Lesen Sie bitte über den Parameter **EERP_OUT** in diesem Kapitel.

**SYNCTO** Lesen Sie bitte über den Parameter **EERP_OUT** in diesem Kapitel.

**TCPIPRCV** Maximale Anzahl der (gleichzeitig) voraktivierten `horchenden` (listening) Prozesse für TCP/IP Kommunikation:

0 kein TCP/IP Empfänger wird gestartet

n TCP/IP Empfänger werden gestartet

Standard: **1**

**TIMESTAMP**

Generierung eines Zeitstempels, um Dateien mit dem gleichen Namen zu unterscheiden

- 0 000-999 (dreistelliger Zähler für MS DOS Dateinamen)
  Hinweis: Sind alle Ziffer vegeben, wird 999.nnn wieder beginnend bei 1 angehängt.
- 1 000000-999999 (Zähler)
  Hinweis: Sind alle Ziffer vergeben, wird 999999.nnnnnn beginnend bei 1 angehängt.
- 2 Thhmmss (Uhrzeit)
- 3 Dyymmdd.Thhmmss (Datum und Uhrzeit)
- 4 Thhmmssmsms (Datum und Zeit mit Angabe der Millisekunden)

Standard: 2 (nur Uhrzeit)

**TMAXCON**

Maximale Anzahl von gleichzeitig laufenden Operator-Konsolen

0 unbegrenzt

Standard: **0**

**TRACEALWAYS-TOFILE**

Dieser Parameter ermöglicht, dass alle Verbindungsdaten in eine Trace-Datei protokolliert werden. Die Trace-Dateien werden im Verzeichnis `$RVSPATH/tracedir` unter dem Namen `traceXXX.log` angelegt. `XXX` steht für die Nummer des rvscom-Prozesses, die auch in der MonitorLog-Datei `$RVSPATH/db/rlog.log` mitprotokolliert wird. Mögliche Werte:

1 aktiviert, 0 ausgeschaltet.

Standard: **0**

**TRACECONNERR**

Jeder Verbindungsfehler wird in eine Trace-Datei mitprotokolliert. Siehe auch Parameter `TRACEALWAYSTOFILE`. Mögliche Werte:

1 aktiviert, 0 ausgeschaltet.

Standard: **0**

**TRACEMAXITEM**

Maximale Anzahl der Zeilen (Records) im Speicher.

Standard: 1024 records

**TRACEMAXSIZE**

Maximale Größe des Traces im Speicher (Einheit:kB).

Standard: 1024 kB.

**TRACEOFF**

Entscheidet, ob die Traces im Speicher geschrieben werden oder nicht. Mögliche Werte:

- **0**: Traces werden im Speicher geschrieben
- **1**: keine Traces werden im Speicher geschrieben.

Standard: **1**

**TSTODPRCT**

Prozentanteil der fehlerlosen Rücksendungen vom Odette Simulationsprogramm, wenn rvs® im Testmodus läuft; **-1** fordert 'prompten' nach den Rückgabewerten

Standard: **90**

**USEPKI**

rvsX kann öffentliche Schlüssel von Partnern via LDAP-Schnittstelle von einer PKI (Public Key Infrastructure) beziehen. Wie Sie dies konfigurieren können, lesen Sie bitte im Kapitel 8.4.

Mögliche Werte:

- Y (YES): An Stelle der rvsX-internen Schlüsselverwaltung soll eine externe LDAP-Schnittstelle für die Verwaltung der öffentlichen Schlüsseln verwendet werden.
- N (NO): rvsX-interne Schlüsselverwaltung soll benutzt werden.

Default: **USEPKI=N**

**VDSNCHAR**

erlaubte Zeichen für VDSN (virtuellen Dateinamen) in einer Odette-Übertragung:

- **ALL**: keine Beschränkung
- **OFTPUNIXS**: alle Großbuchstaben, Ziffern und die folgenden Sonderzeichen: **. -**
- **UNIX**: alle Buchstaben, Ziffern und die folgenden Sonderzeichen: # _ **- +** .
- **ODETTE**: alle Großbuchstaben, Ziffern und die folgenden Sonderzeichen: **() - . / &^**
- **CHECK_RE**: wie **ALL**, es ist jedoch erforderlich, das ein residenter Empfangeintrag (`RE`) vorhanden ist.

Standard: **ODETTE**

**VFTYP**

Art wie die Dateien (mit festem oder variablem Format) vor der Übertragung konvertiert werden.

**V** internes rvs® Format, brauchbar nur für Dateien im Format Fest und Variabel (die Dateien liegen in diesem Format schon vor oder werden vor der Übertragung mit dem Programm `rvsut2fv.exe` konvertiert).

**S** Format von ft-SINIX, brauchbar nur für ft- SINIX

**T** Textformat, die Datei liegt im Textformat vor. Textformat bedeutet, dass die zu sendene Datei nur aus ASCII-Zeichen bestehen darf und jede Zeile mit Zeilenwechsel ('carriage return' und 'line feed' (Windows) oder nur 'line feed' (UNIX)) endet. Jede Zeile wird in einen Ausgabesatz konvertiert. Die Satzlänge wird im Parameter **MAXRECL** festgelegt.

Empfang: Dateien im Format **Fest** oder **Variabel** können bei Empfang auch als Textdatei abgelegt werden. Dies wird über den Parameter **VFTYP** gesteuert. **VFTYP=T** bedeutet, dass empfangene Dateien im Format **F** oder **V** so abgelegt werden, dass am Satzende ein Zeilenwechsel eingefügt wird. Es ist möglich, dies auch im residenten Empfangseintrag zu steuern, indem der Parameter **VFTYP** auf **T** gesetzt wird.

Senden: Dateien können auch im Format **Fest** oder **Variabel** mit der Kommandozeilen-Schnittstelle (`rvsbat`) versendet werden, ohne dass sie zuvor mit dem Utility `rvsut2fv.exe` konvertiert wurden. Dazu kann man mittels des Parameters **MAXRECL** zusätzlich die Satzlänge angeben. Für Dateien im Format **Fest** ist dies die Satzlänge jedes Satzes bis zum Zeilenwechsel (CR/LF bei MS Windows, LF bei UNIX Systemen). Bei Dateien im Format **Variabel** handelt es sich um die maximale Satzlänge. Voraussetzung ist, dass die Dateien in dem entsprechenden Format vorliegen, anderenfalls kommt es zu Übertragungsfehlern. Leerzeilen werden durch einen Satz mit genau einem Leerzeichen ersetzt.

Um Dateien im Format **Fest** oder **Variabel** mit der Kommandozeilen-Schnittstelle (`rvsbat`) versenden zu können, gibt es folgende optionale Parameter für die Erzeugung eines Sendeeintrages:

**VFTYP** legt fest, ob Dateien im Format **F** oder **V** nach der oben beschriebenen Art versendet werden sollen: **T** - Datei liegt im Text-Format vor und wird nach der neuen Methode behandelt; Standard: **V** - Verhalten wie bisher, Dateien im Format **F** oder **V** sind mit `rvsut2fv.exe` formatiert.

**MAXRECL** maximale Satzlänge, wenn **VFTYP=T** gesetzt ist (Beispiele zu diesem Thema befinden sich im

Referenzhandbuch; Kapitel über `rvsbat` (Kommando `SEND`)).

**VFTYP=X** bedeutet, dass Textdateien automatisch beim Versenden in das Host-Format Fixed/Variable mit dem Tool `rvsut2fv` umgewandelt werden. Dabei muss die Textdatei nicht die gewünschte Zeilenlänge besitzen; dies wird vom Tool `rvsut2fv` erledigt. Folgende Parameter sind anzugeben: **VFTYP**, **FORMAT** und **MAXRECL** (Beispiel: `VFTYP=X FORMAT=F MAXRECL=80`).

**VFTYP=U** bedeutet, dass unstrukturierte (binäre) Dateien automatisch beim Versenden in das Host-Format Fixed/Variable mit dem Tool `rvsut2fv` umgewandelt werden. Dabei muss die unstrukturierte Datei nicht die gewünschte Zeilenlänge besitzen; dies wird vom Tool `rvsut2fv` erledigt. Folgende Parameter sind anzugeben: **VFTYP**, **FORMAT** und **MAXRECL** (Beispiel: `VFTYP=U FORMAT=F MAXRECL=80`).

**VFTYP=D** bedeutet, dass Dateien im Format **Fest** als Binärdateien übertragen werden und Dateien im Format **Variabel** als Textdateien übertragen werden.

Standard: **V** (rvs® internes Format)

**XMCREATE**

Generierung von Log-Nachrichten mit ausführlicher Information über die Art der Sendung, Ausgangspunkt und Zielort nach jedem erfolgreichen Sende- oder Empfangsprozess.

Standard: **1** (eingeschaltet)

**XMTTIMEOUT**

Dieser Parameter ist nur im Zusammenhang mit rvs® Data Center von Bedeutung. Timeout für rvs®-MasterTransmitter..

Standard: **700** (Sekunden).

## 8.2 Leitungsfehlerprotokolle im Speicher

rvs® bietet ab Version 4.0 die Möglichkeit, Leitungsfehlerprotokolle im Speicher abzulegen. Diese Funktion ermöglicht eine bessere Diagnose von Fehlern, die auf der Leitungsebene ab und zu auftreten.

Alle ankommenden und abgehenden Daten werden in einem Puffer im Speicher mitgeloggt.

Die Größe des Puffers ist durch die folgenden neuen rvs®-Parameter beeinflussbar: TRACEMAXITEM und TRACEMAXSIZE.

TRACEMAXITEM ist die Anzahl der Zeilen (Records) im Speicher, wohingegen TRACEMAXSIZE die maximale Größe des Protokolls im Speicher ist (Einheit: kB). Standardwert für TRACEMAXITEM: 1024 Zeilen. Standardwert für TRACEMAXSIZE: 1024 kB.

Um alle Verbindungsdaten in eine Protokolldatei zu schreiben, muss der neue Parameter TRACEALWAYSTOFILE aktiviert sein (=1). Schwerwiegende Verbindungsfehler werden immer (auch wenn der Parameter TRACEALWAYSTOFILE nicht aktiviert ist; =0) in eine Protokolldatei geschrieben.

Die Protokolldateien entstehen dann im Verzeichnis $RVSPATH/tracedir unter dem Namen traceXXX.log (XXX steht für die Nummer des rvscom-Prozesses, die auch in der MonitorLog-Datei $RVSPATH/db/rlog.log mitprotokolliert wird).

Der Parameter TRACECONNERR (wenn er aktiviert ist: =1) bewirkt, dass jeder Verbindungsfehler in eine Trace-Datei mitprotokolliert wird.

Die Protokollierung im Speicher ist über den Parameter TRACEOFF abschaltbar. Mögliche Werte: 0 = Protokolle werden geschrieben; 1 = keine Protokolle im Speicher. Diese Möglichkeit ist besonders aus Performanzgründen von Bedeutung, da bestimmte Fehler wegen ihrer Häufigkeit zu überlangen Protokollen führen können.

## 8.3 Robustheit von rvs® bezüglich Speicherplatz

rvs® bietet ab Version 4.05 die Möglichkeit der Speicherplatzüberprüfung aller relevanten rvs®-Verzeichnisse. Dadurch wird die Robustheit von rvs® in Bezug auf Speicherplatzprobleme deutlich erhöht. Mit dieser Funktionalität wird gewährleistet, dass ein rvs®-Operator rechtzeitig auf Speicherplatzmangel reagieren kann.

Im Falle von Speicherplatzproblemen erscheint im Monitor-Log eine Meldung, die besagt, in welchen rvs®-Verzeichnissen nicht genügend Speicherplatz vorhanden ist und wie viel noch benötigt wird. Gleichzeitig wird ein Skript aufgerufen, das Sie Ihren eigenen Bedürfnissen anpassen können.

Je nach Grad des Speicherplatzmangels gibt es die Möglichkeit eine Warnung oder einen Fehler auszugeben. Nach einem Fehler wird zusätzlich noch rvs® gestoppt.

Beispiel für Warnung:

```
W:<DISKSPACEWARN> Plattenplatz auf /home/tr08/rvs/db, /home/tr08/rvs/temp,
/home/tr08/rvs/tracedir, /home/tr08/rvs/usrdat/, /home/tr08/rvs/temp/in,
/home/tr08/rvs/temp/out, /home/tr08/rvs/temp/temp niedrig, benoetigt 11021504 kB , zur
Verfuegung stehen 8610080 kB.
```

Beispiel für Fehler:

```
E:<DISKSPACEERR>Plattenplatz auf /home/tr08/rvs/db, /home/tr08/rvs/temp,
/home/tr08/rvs/tracedir, /home/tr08/rvs/usrdat/, /home/tr08/rvs/temp/in,
/home/tr08/rvs/temp/out, /home/tr08/rvs/temp/temp extrem niedrig, benoetigt 11021504 kB , zur
Verfuegung stehen 8609952 kB.
```

Nach einem Fehler wird rvs® gestoppt:

```
A:  <OK_READ         >  [rvsstop] stop rvs=force
I:  <OK_CMD_DONE     >  [rvsstop]  'stop' beendet.
I:  <XMT_STOP_NORMAL >  Master Transmitter wird normal gestoppt.
I:  <RVS_TERMINATION >  RVS Monitor wurde beendet. (Endecode: 4).
```

Im Falle einer Warnung wird das Skript `diskspacewarn.sh` bzw. `diskspacewarn.bat` und im Falle eines Fehlers das Skript `diskspaceerr.sh` bzw. `diskspaceerr.bat` aufgerufen. Beide Skripte sind als Beispiele im Verzeichnis `$RVSPATH/system` zu finden und können Ihren Bedürfnissen entsprechend angepasst werden. Diese Skripte müssen im Verzeichnis `$RVSPATH/system` gespeichert werden und wie genannt heißen.

**Hinweis**: Skripte für UNIX-Plattformen haben die Endung `.sh` (Shell-Dateien), wohingegen Windows-Skripte die Endung `.bat` (Batch-Dateien) haben.

Beispiel `diskspaceerr.sh`:

```
echo "1 >$1<  2 >$2< 3 >$3<" >> $HOME/diskerr.log
echo "DISKSPACEERR : low disk space on >$1< ( needed $2 kb, available $3
kb)" >> $HOME/diskerr.log
rvsstop -f
```

In diesem Beispielskript wird die Log-Meldung zusätzlich in eine Datei (`$HOME/diskerr.log`) ausgegeben und gleichzeitig wird rvs® gestoppt.

Wie eine Warnung definiert wird (wie viel freier Speicherplatz auf rvs®-Verzeichnissen vorhanden sein muss) lässt sich mit dem Parameter `DISKSPACEWARN` konfigurieren (Siehe Tabelle am Ende dieses Kapitels), wohingegen ein Fehler durch den Parameter `DISKSPACEERR` festzulegen ist.

Folgende rvs®-Verzeichnisse werden in regelmäßigen (durch den Parameter `DISKSPACEDELAY` konfigurierbaren, siehe Tabelle unten) Abständen überprüft: `DB`, `TEMP`, `TRACEDIR`, `USRDAT`, `SPINDIR`, `SPOUTDIR` und `SPFILESDIR`.

Beim Dateiempfang wird jetzt bei Sessionaufbau, wenn der Speicherplatz im Verzeichnis `$RVSPATH/temp` nicht ausreicht, die Log-Meldung „`Datei zu gross`“ ausgegeben. Wie viel Speicherplatz vorhanden sein muss, um einen erfolgreichen

Empfang zu ermöglichen, können Sie mit dem Parameter `DISKSPACEREJECT` einstellen.

Beispiel (Senderseite):

```
A: 2006/03/15 14:11:52  <RPS_TERMINATION >  SK(115) Das Senden des
Kommandos wurde mit dem Returncode 4 beendet (Uebertragung vom Nachbarn
zurueckgewiesen (SFNA oder EFNA))
O: 2006/03/15 14:11:53  <DISCONNECT      >  Verbindung als Sender zu
Station 'WOB_AIX' abgebaut.
I: 2006/03/15 14:12:20  <SERR_ODETTE     >  rpm: Odette-Uebertragung fuer
SK(115) von Knoten WOB_AIX zurueckgewiesen. Uebertragungsbeginn abgelehnt -
Wiederanlauf erlaubt : Datei zu gross
```

Beispiel (Empfängerseite):

```
E: <SHOW_CAUSE> Fehler bei Start oder Ende des Dateitransfers (CAUSE=6):
SFNA: Datei zu gross
O: <DISCONNECT> Verbindung als Empfaenger zu Station 'DIEXP' abgebaut.
```

Hier ist ein Überblick der neuen rvs® Parameter bezüglich der Robustheit vom Speicherplatz:

| Parameter | Bedeutung |
|---|---|
| **DISKSPACEERR** | Anzahl von Kilobytes in jedem zu überprüfenden rvs®-Verzeichnis, die frei sein muss. Ein Unterschreiten bewirkt Aufruf des Skripts `diskspaceerr`.<br>Standard: 1000 (ca. 1 MByte) |
| **DISKSPACEWARN** | Anzahl von Kilobytes in jedem zu überprüfenden rvs®-Verzeichnis, die frei sein muss. Ein Unterschreiten bewirkt Aufruf des Skripts `diskspacewarn`.<br>Standard: 5000 (ca. 5 MByte) |
| **DISKSPACEDELAY** | Anzahl von Sekunden, bevor bei `DISKSPACEWARN` die Log-Meldung erscheint und das Skript `diskspacewarn` erneut aufgerufen wird.<br>Standard: 600 Sekunden (10 Minuten) |
| **DISKSPACEREJECT** | Anzahl von Kilobytes, die imVerzeichnis `$RVSPATH/temp` frei sein müssen, damit Dateien empfangen werden können.<br>Standard: 5000 (ca. 5 MByte) |

## 8.4 rvs®-PKI-Anbindung

In einer PKI werden öffentliche Schlüsssel verwaltet und Zertifikate für sie ausgestellt. Der Zugriff auf öffentliche Schlüssel von Partnern erfolgt via LDAP-Schnittstelle.

Hinweis: LDAP ist ein Netzwerkprotokoll, welches die Kommunikation zwischen einem LDAP-Client und einem LDAP-DirectoryServer (Verzeichnis-Server) regelt. Das Protokoll bietet folgende Funktionen: Anmeldung vom Client am Server, Suchabfrage bzgl. der im Verzeichnis (directory) gespeicherten Informationen und Modifikation von Informationen. Bezogen auf PKI: im LDAP-DirectoryServer werden die Zertifikate (öffentliche Schlüssel) verwaltet. Die öffentlichen Schlüssel werden für die Verschlüsselung benötigt.

Folgende Funktionalitäten bietet die rvs®-PKI-Anbindung:

- Beziehen von öffentlichen Schlüsseln für Partnerstationen via LDAP-Schnittstelle von einer PKI (Public Key Infrastucture).
- Prüfen der öffentlichen Schlüsseln mit OCSP (Online Certificate Status Protocol)
- Übertragung der Informationen zum OCSP-Server mit HTTP

Die rvs®-PKI-Anbindung wird mit dem Parameter `USEPKI` aktiviert. Dieser Parameter kann stationenbezogen für einzelnen Stationen oder global für alle Stationen konfiguriert werden.

**rvsX**

Stationenbezogen: in der Stationstabelle (Datei `$RVSPATH/init/rdstat.dat`) bei den ODETTE-Parametern (Tabelle OP, siehe rvsX Benutzerhandbuch, Kapitel 3.1.6 „Odette-Parameter“)

Global als rvsX-Parameter für alle Stationen in der Datei `$RVSPATH/init/rdmini.dat`.

```
setparm USEPKI=Y
```

**rvsXP**

Stationenbezogen: `rvsXP Administrator -> Stationen -> ODETTE Parameter -> PKI`

Global: `rvsXP Administrator -> Parameter -> USEPKI`

Die Konfiguration der PKI-Anbindung erfolgt über die Datei `$RVSPATH/init/rvspki.dat`

Wenn sie den Parameter USEPKI aktiviert haben (Y=YES), müssen Sie in der Datei $RVSPATH/init/rvspki.dat die Parameter, die für den Zugriff auf eine PKI (Public Key Infrastructure) notwendig sind, konfigurieren.

Beispiel ($RVSPATH/init rvspki.dat):

```
[PKI]
#Host name or IP address of LDAP directory server
DirectoryServer=17.145.32.89

#IP port of LDAP directory server, 389 is used as default, if left empty
DirectoryServerPort=

#User name and password of LDAP directory server,
#anonymous access is used if left empty
DirectoryServerUser=
DirectoryServerPw=

#LDAP version of LDAP directory server, 3 is used as default, if left empty
DirectoryServerLDAPVer=

#dn of LDAP search starting point, from special to global property
LDAPSearchBase=ou=Odette,o=BTGK,dc=BT,dc=hgf,dc=com

#LDAP search filter part 1, Odette ID of the partner station is part 2
LDAPFilterPart1=cn=BT_OFTP

#URL of OCSP server
OCSPServerURL=http://ocsp.btgk.de

#Options
#Skip OCSP check: Options=1
#Ignore OCSP check result and errors: Options=2
Options=2

#Location and file name of the private key file
PrivateKey=/rvs/init/btgk.pri

#Service entries for support purposes only
#PKIStr1=
#PKIStr2=
TestMode=0

[Issuer]
#Locations and file names of certificate issuers' certificates
<CA #1 name>=<path and file name>
<CA #n name>=<path and file name>
```

Die Pflichtparameter für die Anbindung an die PKI sind (fett gedruckt im Text): DirectoryServer, LDAPSearchBase, LDAPFilterPart1 und PrivateKey.

In der folgenden Tabelle werden die Parameter aus der Datei rvspki.dat erklärt:

| Parameter | Bedeutung |
|---|---|
| `DirectoryServer` | Hostname oder die IP-Adresse des LDAP-DirectoryServers |
| `DirectoryServerPort` | Port des LDAP-DirectoryServers<br>Standard: 389 |
| `DirectoryServerUser` | Benutzername auf dem LDAP-DirectoryServer |
| `DirectoryServerPw` | Passwort des Benutzers auf dem LDAP-DirectoryServer |
| `DirectoryServerLDAPVer` | Version des LDAP-Protokolls auf dem LDAP-DirectoryServer<br>Standard: 3 |
| `LDAPSearchBase` | Bezeichnet die Struktur im Directory (Verzeichnis), in der mit der Suche nach Objekten begonnen wird. Dabei steht `ou` für OrganizationalUnit, o für Organization and `dc` für DomainComponent. |
| `LDAPFilterPart1` | LDAP-Suchfilter: z.B. Für die VW-PKI-Anbindung ist der erste Teil des Filters `cn=VW_OFTP`; der zweite ist OdetteID der jeweiligen Station. |
| `OCSPServerURL` | Hier soll die URL des OCSP-Servers angegeben werden.<br>OCSP steht für Online Certifikate Status Protocol. Mit Hilfe eines OCSP-Servers werden die Zertifikate zeitnah (online) auf Ihre Gültigkeit überprüft. Es wird überprüft, ob ein Zertifikat unbekannt, noch gültig oder gesperrt ist. |

| | |
|---|---|
| `Options` | Dieser Parameter bezieht sich auf die OCSP-Überprüfung. Mögliche Werte sind:<br>• 0: OCSP-Überprüfung wird nicht ignoriert.<br>• 1: OCSP-Überprüfung wird komplett ignoriert.<br>• 2: OCSP-Fehler werden gemeldet, führen aber nicht zum Abbruch.<br>Es empfiehlt sich den Parameter `Options` auf 2 zu setzen.<br>`Options` 0 kann unter Umständen bedeuten (wenn z.B. der OCSP-Server nicht erreichbar ist), dass die Verschlüsselung und Entschlüsselung in dieser Zeit, gar nicht funktioniert. |
| `PrivateKey` | Pfad und Dateiname des privaten Schlüssels |
| `ServiceEntries (TestMode)` | Nur für Testzwecke mit T-Systems Kundendienst |
| `[Issuer]: Location and file names of certificate issuers' certificates` | Pfad und der Dateiname der Zertifikate des Zertifikatsherausgebers |

## 8.5 Beschreibung ausgewählter rvs® Parameter

rvs® enthält eine Reihe von optionalen und sicherheitsbezogenen Funktionen, die Sie vielleicht nicht alle (oder die ganze Zeit) in Ihrer Installation brauchen werden. Wenn sie aktiviert sind, verbrauchen diese Funktionen Rechnerressourcen (Prozessorzeit und Festplattenzugang) und können so die Leistung der rvs® Komponenten erheblich beeinflussen.

Lassen Sie uns z.B. die Übertragung großer Datenpakete genauer betrachten. Um in der Lage zu sein, die Übertragung am Unterbrechungspunkt fortzuführen, ohne immer wieder am Dateianfang zu beginnen, schließt der Empfänger das eingehende Datenpaket in regelmäßigen Zeitabständen. Beide, der Sender und der Empfänger speichern die Zahl der übertragenen Bytes oder Aufzeichnungen in der Datenbank. Die Frequenz dieser Aktionen wird in den Parametern **SENDBLOCKS** und **RECVBLOCKS** festgelegt.

Das erneute Öffnen und Positionieren eines großen Datenpakets erfordert eine beträchtliche Zahl von Zugangsoperationen auf die Festplatte und ist deshalb sehr zeitaufwendig[1]. Wenn Sie sehr stabile Kommunikationsleitungen haben, werden Sie diese Parameter auf sehr hohe Werte einstellen wollen, so dass die rvs® Neustart-Funktionen praktisch ausgeschlossen werden.

Wenn andererseits Ihre Leitungen alle paar Minuten unterbrochen werden, wird es sicher Ihr Wunsch sein, dass keine wiederholte Übertragung von schon gesendeten Daten stattfindet. Beachten Sie, dass ein großer Wert von **SENDBLOCKS** auch die Zeit verlängert, nach welcher die Sender geschlossen werden, wenn der Monitor mit dem Kommando `stop rvs=force` beendet wird.

Die Standard-Einstellungen sind so gewählt, dass rvs® sicher und mit den meisten Optionen aktiviert arbeiten wird.

`$RVSPATH`[2]`/system/rdmini.dat` enthält eine Liste der rvs® Parameter und ihrer Standardwerte, in derselben Reihenfolge, in der sie in diesem Kapitel behandelt sind. Verwenden Sie diese Datei, um die Parameter und ihre Standardwerte mit dem Operator-Kommando wiederherzustellen, wenn der rvs® Monitor im Betrieb ist.

Um sicherzustellen, dass der rvs® Monitor immer mit Ihrem lokalen Parameterset arbeitet, kopieren Sie die Datei `rdmini.dat` in die `$RVSPATH/init/defparms.dat`, passen Sie sie an und fügen Sie zu der Liste der Initialisierungskommandos des rvs® Monitors in `$RVSPATH/init/rdmini.dat`. die folgende Zeile hinzu:

```
OPCMD DSN=$RVSPATH/init/defparms.dat
```

## 8.6 Der Parameter STATISTICS

Der Parameter **STATISTICS** steuert das Anlegen der Statistik-Log-Datei `$RVSPATH/db/rlstat.log`, welche die Einträge über gesendeten und empfangenen Dateien enthält. Diese Datei kann für Archivzwecke verwendet werden.

Der Parameter **STATISTICS** soll wie alle anderen globalen rvs®-Parameter entweder in der Datei `$RVSPATH/init/rdmini.dat` gesetzt werden oder nur für eine Session über die rvs®-Konsole verändert werden (siehe Kapitel 8).

Er kann die Werte von 0 bis 7 annehmen.

[1] Die Übertragungszeit für ein 4,5 MB Datenpaket zwischen zwei OS/2 Knoten wurde durch das Ändern dieser Parameterwerte von **10** auf **10000** um den Faktor 10 reduziert (von mehr als einer Stunde auf einige Minuten).

Bei **STATISTICS=0** wird keine Statistik-Log-Datei angelegt.

**STATISTICS=1** bedeutet, dass die Datei `rlstat.log` eine Zeile für jede gesendete oder empfangene Datei mit Dateinamen, Datum, Uhrzeit und Sender/Empfänger-ID enthält.

Beispiel:

```
RCV XPSKK 2005/02/14 08:59:14
/home/skk/rvs/global/usrdat/RVSENVALT.DAT
```

| Wert | Bedeutung |
|---|---|
| RCV | Richtung: RCV (receive=Empfang)<br>Möglich ist auch:<br>SNT (send=senden) |
| XPSKK | Richtung RCV: StationsID des Empfängers; Richtung SNT: StationsID des Senders. |
| 2005/02/14 08:59:14 | Datum und Uhrzeit: Empfang: Zustellung der Datei ins rvs®-Verzeichnis usrdat, Senden: Bereitstellung des Sendeauftrags in die rvs®-Datenbank. |
| /home/…/RVSENVALT.DAT | Pfad (lokal) |

**STATISTICS=2** generiert dieselbe Datei, aber mit ausführlicheren, zusätzlichen Informationen wie z.B. Übertragungs-Dateiname (virtueller Dateiname VDSN), Dateigröße und Kommando-Nummer von SE, SK (Senden) oder IE, IZ (Empfang).

**Beispiel**:

```
RCV XPSKK 31857 31860 2005/02/25 14:47:51 RDSTAT.DAT
/home/skk/rvs/global/usrdat/RDSTAT.DAT 666
```

| **Wert** | **Bedeutung** |
|---|---|
| `RCV` | Richtung: `RCV` (receive=Empfang) Möglich ist auch: `SNT` (send=senden) |
| `XPSKK` | Richtung `RCV`: StationsID des Empfängers; Richtung `SND`: StationsID des Senders. |
| `31211 31215` | Kommandonummer (`IE` und `IZ` beim Empfang, `SE` und `SK` beim Senden). |
| `2005/02/14 08:59:14` | Datum und Uhrzeit: Empfang: Zustellung der Datei ins rvs®-Verzeichnis `usrdat`, Senden: Bereitstellung des Sendeauftrags in die rvs®-Datenbank. |
| `RVSENVALT.DAT` | VDSN (virtueller Dateiname für die Übertragung). |
| `/home/…/RVSENVALT.DAT` | Pfad (lokal) |
| `666` | Rictung `RCV`: Anzahl empfangener Bytes; Richtung `SNT`: Anzahl gesendeter Bytes |

**STATISTICS=3** generiert die Datei `rlstat.log` mit denselben Einträgen wie **STATISTICS=1**.

**STATISTICS=4** bewirkt dasselbe wie **STATISTICS=2**, loggt aber ebenfalls gleichzeitig den gerouteten Datentransfer (Ausgangs- und Zielstation).

**Beispiel**:

```
SNT MB01 MBX-MB01 31857 31860 2005/02/25 14:47:51 RTEST
/home/skk/rvs/out/RTEST.DAT 666
```

Im obigen Beispiel wird die Datei `RTEST.DAT` von der Station `MBX` an die Station `MB01` (`MBX-MB01`) gesendet.

**STATISTICS=5** generiert einen noch ausführlichen Output in `rlstat.log` mit zusätzlichen Einträgen wie Dateiformat, Status der Übertragung und Anzahl der Einwahlversuche.

Beispiel:

```
RCV XPSKK 00000319180000031921 - RDSTAT.DAT XPSKK 2005/02/25
14:52:04  2005/02/25 14:52:04
/home/skk/rvs/global/usrdat/RDSTAT.DAT 0 2208 U 0 - - R
```

| Wert | Bedeutung |
|---|---|
| RCV | Richtung, mögliche Werte:RCV (receive=empfangen), SNT (senden). |
| XPSKK | Richtung RCV: StationsID des Empfängers; Richtung SNT StationsID des Senders. |
| 00000319180000031921 | Kommandonummer, 10-stellig (IE und IZ beim Empfang, SE und SK beim Senden). |
| 2005/02/14 08:59:24 | Datum und Uhrzeit: Empfang: Zustellung der Datei ins rvs®-Verzeichnis usrdat, Senden: Bereitstellung des Sendeauftrags in die rvs®-Datenbank. |
| 2005/02/14 08:59:34 | Datum und Uhrzeit der Zustellung der EERP (End-to-End-Response). Nach ODETTE gilt eine Datei als vollständig empfangen, erst wenn die EERP angekommen ist. |
| RVSENVALT.DAT | VDSN (virtueller Dateiname für die Übertragung). |
| /home/…/RVSENVALT.DAT | Pfad (lokal) |
| 2208 | Anzahl übertragener Bytes |
| 2208 | Dateigröße (in Bytes) |
| U | Dateiformat: T = Textdatei; U = unstrukturierte (binäre) Datei; V = variable Satzlänge; F = feste Satzlänge. |
| 0 | Anzahl Sendeversuche |
| - | Status; Möglich ist auch: en (ended) für beendet. |

| | |
|---|---|
| - | der Grund des Löschens (Kommentar, freier Text), wenn mit delcmd spezifiziert. |
| R | Richtung:RCV(receive=Empfang)<br>SNT (send=senden) |

**STATISTICS=6** generiert eine ausführliche Ausgabe über (vom Benutzer) gelöschte Einträge und den Grund des Löschens als Kommentar (wenn er mit delcmd spezifiziert ist).

**STATISTICS=7** bewirkt dasselbe wie **STATISTICS=6**, inklusive Routing.

## 8.7 Sicherheit, Ressourcen-Verbrauch und Leistung

Neben **RECVBLOCKS** und **SENDBLOCKS,** die eben diskutiert wurden, beeinflussen mehrere andere Parameter das Gleichgewicht zwischen Sicherheit, Ressourcenverbrauch und Leistung.

**OCREVAL** (empfohlene Fenstergröße 99) und **OEXBUF** (empfohlene Größe 4096 Bytes) beeinflussen den Overhead, verursacht vom Odette Protokoll. Je höher diese Werte sind, desto geringer ist der Overhead. Sie erhöhen jedoch gleichzeitig den Speicherverbrauch der Sender und Empfänger. Diese Werte können vor dem Start einer jeder Übertragung verhandelt werden, so dass einseitige Änderungen ohne Auswirkung bleiben. Was Sie wirklich bestimmen ist der maximale Speicherplatz, den Sie Odette zur Verfügung stellen.

Das Suchen in einer großen Datenbank dauert erheblich länger als in einer kleinen Datenbank. Eine größere Datenbank enthält jedoch mehr Informationen über durchgeführte Übertragungen. **KEEPDAYS** bestimmt, wieviel Tage Sie die Informationen über durchgeführte oder gelöschte Übertragungen behalten möchten, (falls Sie nicht das Kommando `cleanup days=n` verwenden, das die gewünschte Zeitspanne explizit spezifiziert).

Bei **CMDDELETE = 1** werden alle Einträge physikalisch aus der rvs® Datenbank entfernt, wenn ein Kommando ausgeführt oder (logisch) gelöscht ist. Das reduziert die Größe der Datenbank auf das notwendige Minimum. Wenn Sie diese Option wählen, lassen Sie **XMCREATE** mit seinem Standardwert (**1**). Auf diese Weise werden Log-Nachrichten nach jedem Senden oder Empfangen eines Datenpakets generiert. Um diese Nachrichten sehen zu können, sollten alle Benutzer Zugang zu dem Log-Datensatz ($RVSPATH/db/rlog.log) haben, weil die Dialog-Schnittstelle keine Informationen über abgeschlossene Übertragungen anzeigen kann. Ziehen Sie bei dauerhaften unbetreuten Operationen diese Option in Betracht.

Die Reaktionszeit des rvs® Monitor auf neue Ereignisse wird von **SLEEP** festgelegt. Das beeinflußt z.B. die Reaktionszeit des rvs® Monitor auf ein Operator-Kommando. **SLEEP** ist die Zeitspanne (in Sekunden), für die sich der rvs® Monitor abschaltet, wenn nichts ansteht. Je länger diese Zeitspanne ist, desto weniger beeinflußt sie die anderen Anwendungen, aber Sie müssen auch länger darauf warten, dass der rvs® Monitor Ihr Kommando auszuführen beginnt. Je kürzer Sie die Zeitspanne bestimmen, desto größer ist der (unproduktive) Overhead, der in nicht ausgelasteten Zeiten durch das Durchsuchen der rvs® Datenbank entsteht.

Die Zeitspanne bis der rvs® Monitor eine erfolglose oder abgebrochene Übertragung wieder aufnimmt, wird von den **DTCONNxx** Parametern bestimmt. Je kleiner ihre Werte sind, desto schneller wird die Übertragung gestartet, wenn die Leitung wieder verfügbar ist. Durch die erfolglosen Versuchen bis die Leitung wiederhergestellt ist, wird jedoch um so mehr Rechnerzeit verschwendet.

### 8.7.1 Beschränkung der Anzahl von konkurrierenden Sendern

Wenn Ihr System sehr ausgelastet ist, und Sie wissen, dass Ihre Nachbarn nur ein paar eingehende Anrufe gleichzeitig annehmen können, wollen Sie sicher die Zahl der Sender beschränken, die rvs® gleichzeitig betreiben darf.

Dem Parameter **MAXSENDERS** entnimmt der MasterTransmitter `rvsxmt`, wieviel Sender gleichzeitig in Betrieb sein dürfen. Wenn die Zahl erreicht ist, wartet der MasterTransmitter bis ein Sender beendet wird, bevor er den nächsten Sender startet. Wenn **MAXSENDERS** auf **0** gesetzt ist, wird gar kein Sender gestartet. Das ist nur dann nützlich, wenn die Partnerstation die Verbindung herstellen und die vorbereiteten Datenpakete abholen soll. Verwenden Sie das Kommando `activate`, um Daten zu einer bestimmten Station zu senden, auch wenn **MAXSENDERS** auf **0** gesetzt ist.

### 8.7.2 Beschränkung der Anzahl von konkurrierenden X.25 oder ISDN Empfängern

Sie müssen die Anzahl der gleichzeitig aktiven X.25 und ISDN Empfänger bestimmen. Eine kleine Anzahl eignet sich für geringen Verkehr, eine größere Anzahl ist erforderlich, wenn Sie Daten über mehrere parallele Verbindungen empfangen müssen. Sie können jedoch nicht mehr X.25 Empfänger aktivieren als die Zahl der in Ihrem X.25 Mehrfachkanal vorhandenen virtuellen Kanäle. Bei ISDN können Sie nicht mehr Empfänger starten als die Zahl der verfügbaren B-Kanäle. Weil die Sender auch die virtuellen Kanäle oder bei ISDN die B-Kanäle benutzen, sollte die Zahl der Empfänger auf die Hälfte der verfügbaren Kanäle beschränkt werden.

Dem Parameter **MAXX25RCV** entnimmt der MasterTransmitter `rvsxmt` wieviel Empfänger gleichzeitig im Betrieb sein müssen. Er wird soviel Empfänger voraktivieren, wie in dem Parameter angegeben sind. Wenn ein Empfänger beendet wird, startet der MasterTransmitter einen neuen, der seinerseits auf eingehende Anrufe wartet. Wenn der Parameter **MAXX25RCV** auf **0** gesetzt ist, sind keine

eingehenden X.25 oder ISDN Übertragungen möglich. Er muß auf 0 gesetzt sein, wenn Sie nur SNA-LU 6.2 oder TCP/IP Kommunikation nutzen.

Auf produktiven Systemen müssen Sie zusätzliche Einträge in der X.25 Routing-Tabelle definieren, wenn der Wert von **MAXX25RCV** höher als 1 ist (siehe Kapitel 8.1 "Die rvs® Parameter im Überblick").

### 8.7.3 TCP/IP Empfänger

Wenn Sie über TCP/IP kommunizieren möchten, muß rvs® einen Empfänger starten, der auf eingehende Anrufe wartet. Sie müssen den Wert des Parameters **TCPIPRCV** einstellen. Wenn Sie nur LU 6.2 oder TCP/IP benutzen, setzen Sie **TCPIPRCV** auf 0. Wenn ein TCP/IP Empfänger einen eingehenden Anruf empfängt, startet der MasterTransmitter auf demselben Port einen neuen Empfänger, der auf Anrufe wartet. Die Zahl der Anrufe, die Sie auf jedem Port empfangen können, entspricht den Werten von **MAX_IN**, die für Ihre lokale Station in Ihrer Stationstabelle `$RVSPATH`[3]`/init/rdstat.dat` definiert sind.

### 8.7.4 Optionale Funktionen

Das Bereitstellen der optionalen Dienste verbraucht Zeit und Speicherplatz, aus diesem Grund würden Sie sie sicher abschalten wollen, wenn Sie sie nicht brauchen.

**AECCHECK** ist ein Flag, das den rvs® Monitor anweist, für das aktuell auszuführende Kommando zu überprüfen, ob die Person, die es eingegeben hat, auch dazu berechtigt ist. In einer (zukünftigen) Umgebung mit mehreren Konsolen, kann diese Funktion z.B. verhindern, dass von einer der Konsolen aus der Monitor ausgeschaltet wird. Diese Funktion wird noch nicht voll unterstützt, deshalb sollte **AECCHECK=0** bleiben (ausgeschaltet).

Wenn **BBCREATE** eingeschaltet ist (**BBCREATE=1**), werden Benutzerbenachrichtigungen generiert und zum User Exit `rvsums` geschickt. Das normale Vorgehen ist, diese Nachricht als UNIX Mail zu senden (geben Sie `mail` ein, um die Nachricht zu lesen). **?** verhindert die Generierung von Benutzerbenachrichtigungen.

Für alle Übertragungsversuche werden statistische Aufzeichnungen angelegt, wenn das Flag **SSCREATE** eingeschaltet ist (**SSCREATE=1**). Diese Aufzeichnungen enthalten die Stations-ID der Nachbarstation, die Zeit und den Abschlußcode der versuchten oder der abgeschlossenen Übertragungen. Bei **SSCREATE=0** werden keine solchen Aufzeichnungen angelegt. Zurzeit gibt es noch keine Funktion zur Auswertung dieser Aufzeichnungen.

**XMCREATE** (generiere `xfer` Nachricht) steuert das Anlegen ausführlicher Informationen über die erfolgreichen Übertragungen in die System-Log-Datei

[3] siehe Kapitel 1.1 "Repräsentationsmittel" für eine ausführliche Beschreibung von **$RVSPATH**

`$RVSPATH`[4]`/db/rlog.log`. Bei **XMCREATE=1** (Normalfall) wird in den folgenden Fällen eine Log-Nachricht geschrieben: bei jeder erfolgreichen Übertragung eines Datenpakets zu einer Nachbarstation (noch bevor die Bestätigung eingegangen ist); beim Abschluß eines Sendeeintrags (wenn die Bestätigungen aller Empfänger eingegangen sind) und bei Lieferung eines Datenpakets an einen lokalen Benutzer. Bei **XMCREATE=0** werden keine Log-Nachrichten generiert.

Beim Auftreten von Kommunikationsfehlern kann man hilfreiche Informationen in den Verfolgungsdatensätzen finden, wenn die Werte der Parameter **LITRACELVL** und **ODTRACELVL** größer als 0 sind. Verfolgung (Tracing) kann die Leistung erheblich reduzieren, weil eine große Datenmenge analysiert, formatiert und in die Verfolgungsdatei geschrieben werden muß. Im Normalbetrieb sollte die Verfolgung ausgeschaltet sein, d.h. beide Parameter sollten auf **0** gesetzt sein.

**CNSMSGS** steuert, welche Log-Nachrichten als Echo auf der Operator-Konsole ausgegeben werden. Alle Nachrichten, deren Code-Buchstabe im (Character String) Wert von **CNSMSGS** enthalten sind, werden auf der Konsole ausgegeben (alle Nachrichten werden immer geloggt, unabhängig vom Wert des **CNSMSGS** Parameters). Die zusätzlichen Nachrichtentypen **O** (Odette), **L** (Leitungstreiber) und (**+**) (für lange Nachrichten) können auch verwendet werden.

### 8.7.5 Interne Parameter

Mehrere Kommandos werden für den internen Betrieb des rvs® Monitors benutzt. **Beachten** Sie, dass Sie ihre Werte mit `listparm` auflisten können, dürfen Sie jedoch niemals mit `setparm` ändern!

Interne Parameter sind:

| | |
|---|---|
| **FORCEDEND** | gesetzt durch `stoprvs=force`, und |
| **INITCMDS** | gesetzt durch **/i**-Kommandozeilen-Flag |

[4] siehe Kapitel 1.1 "Repräsentationsmittel" für eine ausführliche Beschreibung von **$RVSPATH**

# III. Hilfswerkzeuge

Alle nützlichen und wichtigen Hilfswerkzeuge (tools) des rvs®-Systems werden in diesem Kapitel beschrieben.

## 9 Liste aller rvs® Hilfswerkzeuge

Diese Liste gibt Ihnen einen Überblick über sämtliche rvs® Hilfswerkzeuge und ihre Verfügbarkeit auf den verschiedenen Plattformen.

- `rvsjs` (Windows und UNIX)
- `rvswrdstat` (Windows und UNIX)
- `rvsut2fv` (Windows und UNIX)
- `rvsap` (Windows, AIX, IRIX und Solaris)
- `rvsrii` (Windows und UNIX)
- `rvsie` (Windows und UNIX)
- `rvsbackup` (nur UNIX)
- `rvsrestore` (nur UNIX)
- `rvseerp` (Windows und UNIX)
- `rvssend` (nur UNIX)
- `rvssce` (Windows und UNIX)
- `rvscheckdb` (Windows und UNIX)
- `rvshalt` (nur UNIX)

### 9.1 rvsjs

In diesem Kapitel werden die Funktionsweise, Installation und die Konfiguration von rvsjs erklärt.

#### 9.1.1 Einführung: rvsjs als Erweiterung der Batch-Schnittstelle

Über die Batch-Schnittstelle `rvsbat` wird ein Versandauftrag mit folgenden Schritten eingeleitet:

1. Versanddatei bereitstellen (**Beispiel:** `test.L33)`

2. Jobdatei mit `rvsbat`-Befehlen bereitstellen (enthält Versandparameter wie Ziel, OFTP-Namen, Konvertierung). Eine Jobdatei `send.job` könnte folgendermaßen aussehen:

**Beispiel:**
`SEND /C DSN=C:\rvs_out\test.L33 FORMAT=F CODEIN=A (SID=L33 DSNNEW=L33B66.TXT CODEOUT=E)`
In diesem Beispiel wird ein Sendeauftrag erzeugt, um die Datei `test.L33`, die schon im Format Fixed (F) in ASCII-Code (A) vorliegt, an die Station L33 unter dem Namen `L33B66.TXT` zu senden und dabei in EBCDIC-Code (E) zu konvertieren.
**Hinweis**: `rvsbat`-Syntax ist im Kapitel 10 beschrieben.

3. Das `rvsbat`-Kommando mit der Jobdatei `send.job` als Eingabe aufrufen

   **Beispiel:**
   `rvsbat /iC:\senden\send.job`

Dabei muss das `rvsbat`-Kommando auf dem Rechner ausgeführt werden, auf dem rvs® installiert ist.

**Hinweis**: Alle oben aufgeführten Beispiele beziehen sich auf die Windows-Systeme; für UNIX-Systeme lauten die Beispiele gleich, es sollte nur die entsprechende Dateinamensnotation (z.B. `/home/skk/rvs/send/send.job` anstatt `C:\senden\send.job`) angewendet werden.

rvsjs erweitert nun die Batch-Schnittstelle, um zu ermöglichen, Versandaufträge auch über ein Netzwerk zu erzeugen.

### 9.1.2 Arbeitsweise von rvsjs

rvsjs übergibt Jobdateien, die in einem vereinbarten Verzeichnis (Jobverzeichnis) ) abgelegt sind, an `rvsbat` (so dass der dritte Schritt aus dem Beispiel im Kapitel 9.1.1 automatisch von rvsjs ausgeführt wird und der Benutzer die Batch-Schnittstelle `rvsbat` nicht explizit aufrufen muss). Wenn das Jobverzeichnis für Jobdateien im Netzwerk zugreifbar ist, können Sendeaufträge über das Netzwerk an rvs® übergeben werden. Eine solche Konfiguration ist in Abbildung 1 dargestellt:

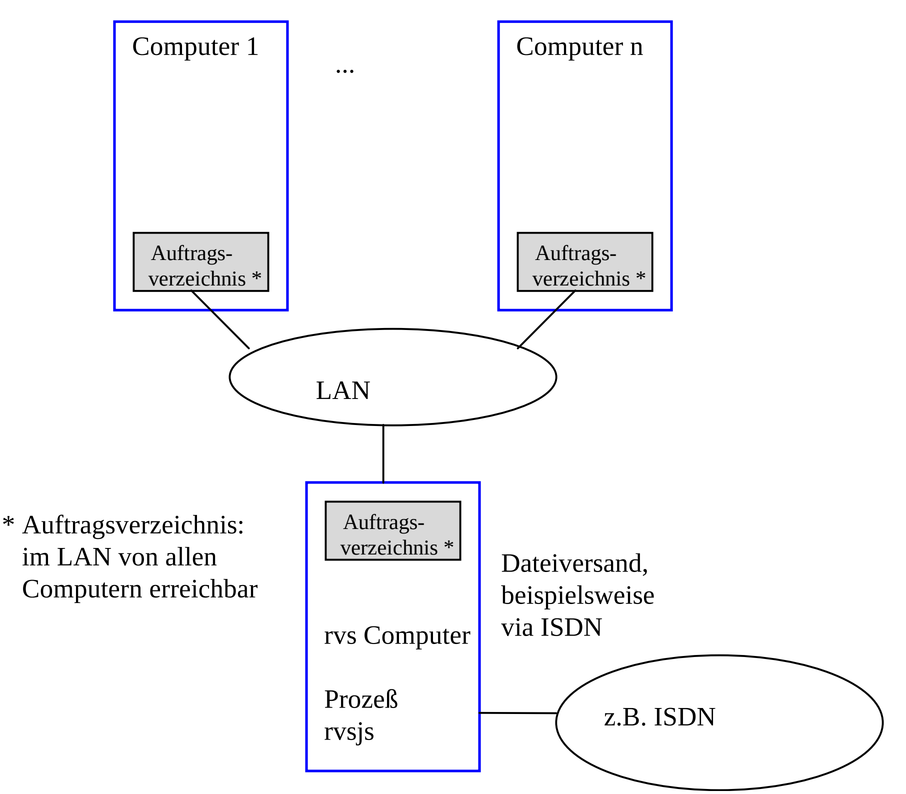



Abbildung 1: Beispielkonfiguration für Übergabe von Versandaufträgen an rvs® über ein Netzwerkverzeichnis

Aus Abbildung 1 ist auch ersichtlich, dass über das Netzwerkverzeichnis Sendeaufträge von mehreren anderen Rechnern (UNIX und/oder Windows) an rvs® übergeben werden können.

rvsjs arbeitet zyklisch mit folgenden Schritten:

- rvsjs überwacht ein oder mehrere Jobverzeichnisse, um Jobdateien für rvs® zu finden. Die Jobdateien werden anhand des (konfigurierbaren) Dateinamens (z.B. `*.job`) erkannt.
- Findet rvsjs eine Jobdatei, benennt rvsjs sie um, um weitere Zugriffe auf sie zu verhindern. Dieser Zwischendateiname hat die Endung `.js`.
- rvsjs startet `rvsbat` (bzw. eine konfigurierbare andere Applikation) und übergibt die Jobdatei mit der Kommandozeile.
- Nach Terminieren von `rvsbat` (bzw. der anderen Applikation) wertet rvsjs deren Rückgabewert aus. Falls der Rückgabewert `0` ist, geht rvsjs davon aus, dass die Ausführung erfolgreich war und löscht die Jobdatei `*.js`. Falls der Rückgabewert ungleich `0` ist, sieht rvsjs den Aufruf als gescheitert an, schreibt den Rückgabewert in die Jobdatei und benennt sie nach `*.err` um. Danach fährt rvsjs wieder mit dem ersten Schritt fort.

Erläuterung zur Jobdatei:

Eine Jobdatei kann neben Sendekommandos beliebige andere (auch mehrere) rvs® Kommandos enthalten, um beispielsweise

- Sendeaufträge zu erzeugen oder zu löschen (die entsprechenden `rvsbat`-Befehle der Batch-Schnittstelle lauten: `SEND /CREATE`, `SEND /DELETE`),
- residente Empfangseinträge und Jobstarteinträge zu erzeugen, zu löschen oder zu modifizieren (`RESENTR /CREATE`, `RESENTR /DELETE`, `RESENTR /UPDATE`; `SENDJOB /CREATE`, `SENDJOB /DELETE`, `SENDJOB /UPDATE`),
- die Stationstabelle zu verändern (`MODST` Befehl),
- rvs® Parameter zu setzen oder aufzulisten (`SETPARM` und `LISTPARM` Befehle).

Näheres über die rvs®-Kommandos können Sie im Kapitel 10 nachlesen.

Beim Start von rvsjs können Sie festlegen, in welchen Zeitabständen die Jobdateien abgearbeitet werden sollen (siehe Kapitel 9.1.5).

In Verbindung mit `rvsbat` ist zu beachten, dass die Jobdatei von rvs® ausgewertet wird. Dies führt dazu, dass Verweise auf Versand-Dateien in der Jobdatei auf die Netzwerksicht des rvs® Rechners bezogen sein müssen.

### 9.1.3 Installation

Dieses Kapitel beschreibt die Installation von rvsjs auf Windows und auf UNIX Systemen.

**Windows**

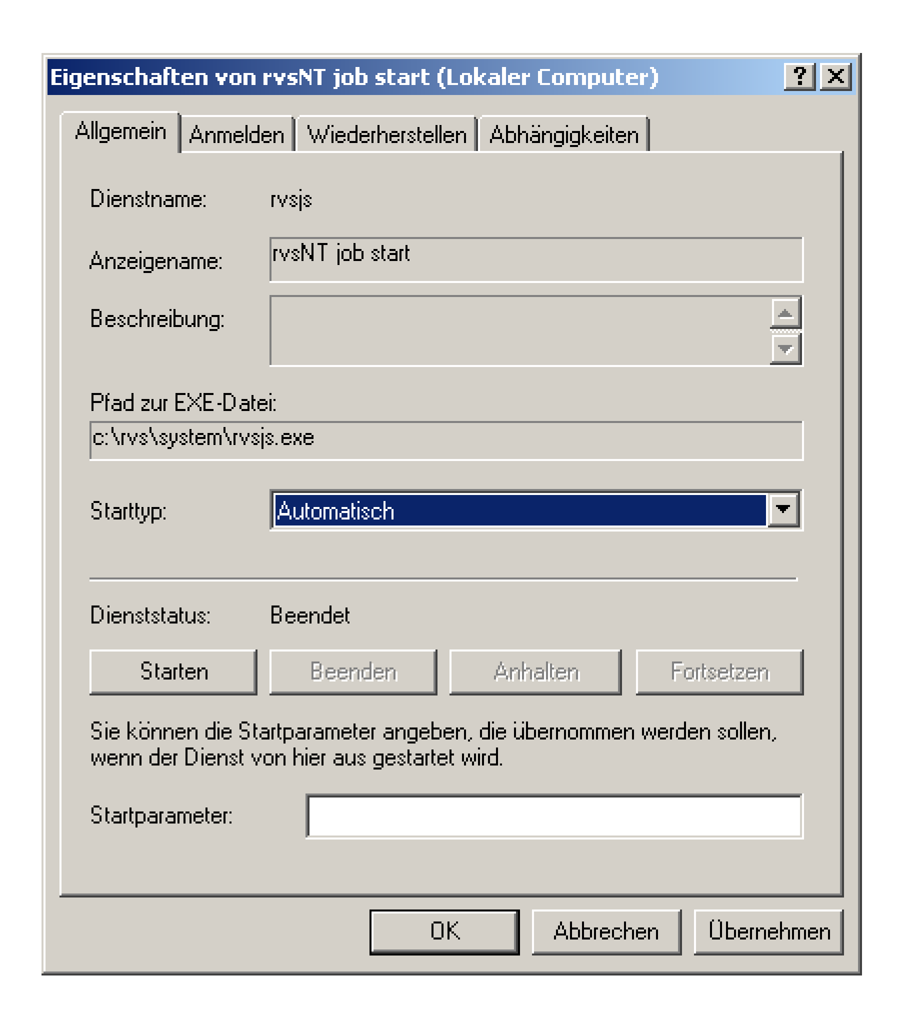
Um rvsjs benutzen zu können, muss sich die ausführbare Datei `rvsjs.exe` im Verzeichnis `$RVSPATH/system` befinden. Installieren als Windows-Dienst kann man es mit dem folgenden Kommando auf der Eingabeaufforderung:

```
rvsjs -i
```

Mit

```
rvsjs -i -s
```

wird rvsjs als Dienst installiert, der mit dem Betriebssystem automatisch gestartet wird. Das Gleiche erzielt man über die graphische Benutzeroberfläche mit `Start → Systemsteuerung → Verwaltung → Dienste → rvs job start`:




Falls rvsjs unter einem bestimmten Benutzerkonto arbeiten soll, so kann dies mit dem Eingabeaufforderungsbefehl:

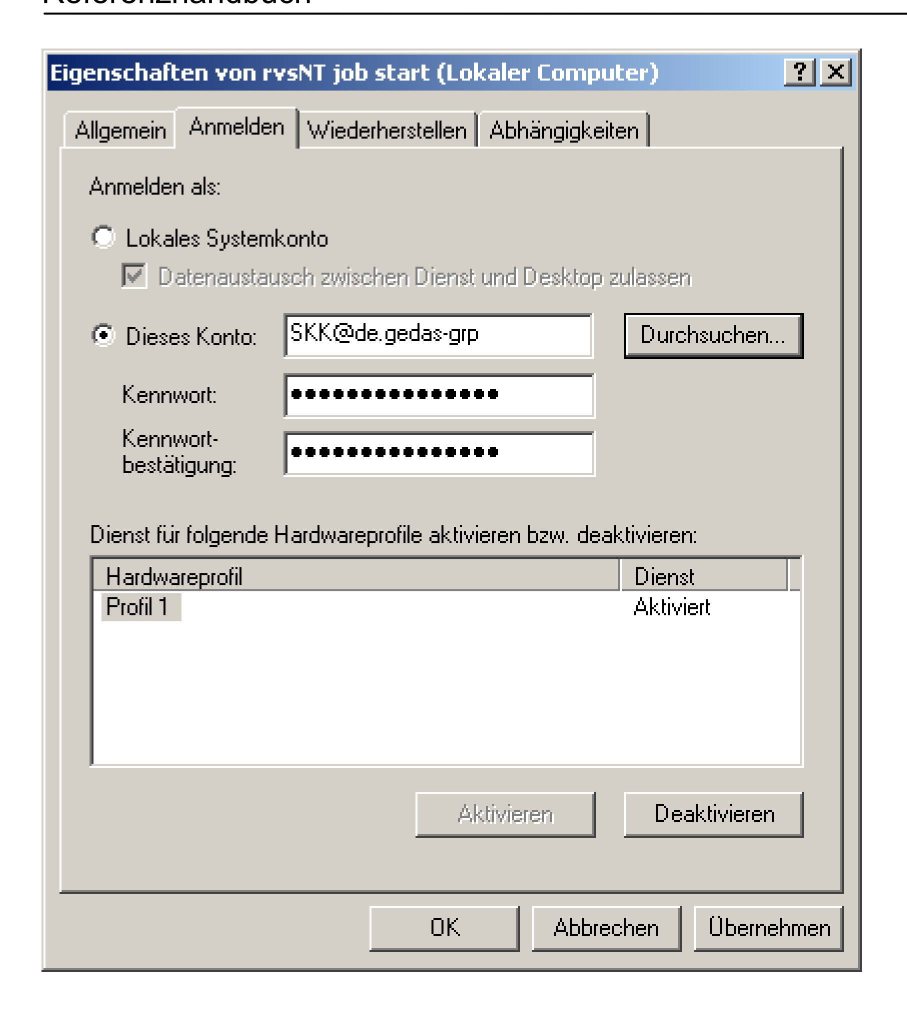
```
rvsjs -i -u <Domänenname\Benutzername> -x <Kennwort>
```

konfiguriert werden. Der Domänenname für den lokalen Rechner ist ‚.‘.

Beispiel:

```
rvsjs -i -u TSYSTEMS\GOR -x gor1?
```

Beispiel auf der Benutzeroberfläche:




**Hinweis:** Wenn rvsjs nicht unter dem Systemkonto arbeitet, sind Ausgaben auf den aktuellen Desktop nicht möglich. Folglich kann rvsjs in diesem Fall nicht benutzt werden, um Applikationen zu starten, die Bildschirmausgaben erfordern.

**rvsjs und „Samba“ Laufwerk**: Seit Samba Version 2.0 unterstützt Samba „directory change notification“ (für rvsjs nötig).

**UNIX**

Die ausführbare Datei `rvsjs` muss sich auch im Verzeichnis `$RVSPATH\system` befinden, aber im Vergleich zu rvsjs in Windows-Systemen sind keine gesonderten Installationsschritte erforderlich.

### 9.1.4 Konfiguration

Dieses Kapitel beschreibt die Konfiguration von rvsjs sowohl über die Kommandozeile als auch über die Konfigurationsdatei `$RVSPATH\init\rvsjs.cnf`.

**Kommandozeile**

Ein Jobverzeichnis (das Verzeichnis, welches rvsjs überwacht) wird an rvsjs mit dem folgenden Kommando

```
rvsjs -p <Auftragsverzeichnis für Jobdateien>
[-a <Applikation>][-j Jobdateimaske>] [-g]
```

auf der Systemebene (Eingabeaufforderung) übergeben.

| Parameter | Beschreibung |
|---|---|
| -p <Jobverzeichnis> | Legt das Verzeichnis fest, welches nach Jobdateien abgesucht werden soll. |
| -a <Jobapplikation> | Setzt den kompletten Dateinamen der Anwendung, die gestartet werden soll, wenn eine Jobdatei gefunden wurde. |
| -j <Jobdateimaske> | Setzt den Filter für den Dateinamen, nach dem gesucht werden soll. Es können die Platzhalter ***** (beliebig viele Zeichen) und **?** (ein Zeichen) eingesetzt werden.<br>**Beispiel**: `*.send` |
| -g | Gibt an, dass rvsjs eine Generationsdatei erzeugen soll, bevor die Jobdatei bearbeitet wird. Dazu wird die Jobdatei mit einem laufenden Zähler nach dem Muster .nnnnn erweitert, wobei nnnnn eine laufende Nummer ist, die mit `.00001` beginnt. Bereits vorhandene Dateierweiterungen werden nicht ersetzt.<br>**Beispiel**: `neu.send.00001`<br>Die Generationsbezeichnung wird immer angehängt. Dateien, die schon eine Generationsbezeichnung besitzen, werden ignoriert. |

Angabe einer Applikation und Jobdateimaske sind optional. Falls nicht angegeben, werden `rvsbat` bzw. `*.job` verwendet.

**Beispiel für Windows:**

```
rvsjs -p C:\rvs_out -j *.snd
```

In diesem Beispiel wird als Jobverzeichnis `C:\rvs_out` und als Maske für die Jobdateien `*.snd` festgelegt. rvsjs wird im Verzeichnis `C:\rvs_out` nach Dateien mit der Endung `.snd` suchen, um sie sofort auszuführen. Als Standard-Applikation wird `rvsbat` verwendet.

**Beispiel für Windows:**

```
rvsjs -p C:\senden -a rvssdat.bat -j *.bar -g
```

Hier wird als Jobverzeichnis `C:\senden` benutzt und als Applikation `rvssdat.bat`. Die Dateien, die im Jobverzeichnis gesucht werden, haben die Endung `.bar` und es wird auch eine Generationsdatei erzeugt. Diese Datei wird auch im Verzeichnis `C:\senden` erzeugt und z.B. `M88L33.bar.00001` heißen. Ihr Inhalt ist identisch mit dem Inhalt der ursprünglichen Datei `M88L33.bar`.

rvsjs kann auch mehrere Jobverzeichnisse (kein Maximum) gleichzeitig bearbeiten. Sie werden durch separate Aufrufe `rvsjs -p...` konfiguriert.

Auf Windows-Systemen gibt die Option `-r` gibt die Möglichkeit, ein nicht mehr benötigtes Jobverzeichnis zu entfernen.

**Beispiel:**

```
rvsjs -r C:\senden
```

**Hinweis:** (Windows-Systeme): Es ist auch möglich rvsjs für den automatischen Start mit Betriebssystem auf der Benutzeroberfläche (Schaltfläche **Startart**) zu konfigurieren (Siehe Installation).

### Konfigurationsdatei `rvsjs`

Nach dem ersten Konfigurationsschritt entsteht im Verzeichnis `$RVSPATH/init` die Konfigurationsdatei `rvsjs.cnf`. Alle Konfigurationsänderungen werden in dieser Datei festgehalten. Sie können diese Datei auch selbst editieren, indem Sie einen einfachen Text-Editor benutzen.

**Windows-Systeme:**

Folgende Syntax soll in der Konfigurationsdatei angewendet werden:

```
<Jobverzeichnis> [-g] <Jobdateimaske> <Jobapplikation>
```

Als Trennzeichen zwischen den Parametern soll ein Leerzeichen oder ein Tabulatorzeichen verwendet werden.

**Beispiel**:

```
c:\out    -g   *vfs*     C:\jobs\sendfile.cmd
```

In diesem Beispiel soll das Jobverzeichnis `C:\out` nach Dateien, die dem Muster `*vfs*` entsprechen, überprüft werden. Wenn solche Dateien vorliegen, soll die

Jobdatei `C:\jobs\sendfile.cmd` ausgeführt werden. Als Trennzeichen wurde das Tabulatorzeichen verwendet.

Beispiel für `sendfile.cmd`:

```
SEND /C DSN=C:\gfd\calc007.txt (SID=O43 DSNNEW=calc7)
```

In diesem Beispiel wurde das `rvsbat`-Kommando `SEND` benutzt, um die Datei `C:\gfd\calc007.txt` an die Station O43 zu versenden.

**UNIX-Systeme**:

Jede Zeile muss in folgendem Format geschrieben sein:

```
<Jobverzeichnis> tab [-g] tab <Jobdateimaske> tab <Jobapplikation>
```

Zwischen den einzelnen Parametern müssen Tabulatoren als Trennzeichen verwendet werden.

Wenn Sie für die beiden optionalen Parameter `<Jobdateimaske>` und `<Jobapplikation>` die Standardeinstellung `*.job` und `rvsbat` wählen, müssen diese Parameter nicht in der Konfigurationsdatei gesetzt werden. Im Falle, dass Sie nur für einen der beiden Parameter die Standardeinstellung wünschen, müssen alle Parameter in der Zeile angegeben werden, da die Reihenfolge der Parameter entscheidend ist.

**Hinweis:** Alle Änderungen in der Datei `rvsjs.cnf` wirken sich erst nach einem Neustart von rvsjs aus. Die aktuelle Konfiguration aus der Datei `rvsjs.cnf` kann auch mit dem Befehl `rvsjs -?` abgefragt werden.

### 9.1.5 Start und Stopp

In diesem Kapitel wird der Start und Stopp auf Windows- und auf UNIX-Systemen beschrieben. Es empfiehlt sich, zuerst rvsjs zu konfigurieren und erst dann zu starten, da rvsjs schon beim Start das Jobverzeichnis nach Jobdateien überprüft.

**Windows**

Starten Sie rvsjs auf Windows-Systemen mit dem folgenden Befehl:

```
rvsjs –t <Aktualisierungsintervall in Sekunden>
```

Mit der Option `–t` legen Sie fest, in welchen Zeitabständen das Jobverzeichnis auf Jobdateien überprüft werden soll.

Beispiel:

```
rvsjs –t 60
```

Das Jobverzeichnis wird periodisch alle 60 Sekunden überprüft.

**rvsjs als Dienst starten**

Der rvsjs-Dienst kann über die grafische Oberfläche von Windows gestartet werden (auch so konfiguriert werden, dass er jedes Mal beim Systemstart zusammen mit Windows gestartet wird, siehe Kapitel 9.1.3).

Auf der Kommandozeile (DOS prompt) oder in batch-Programmen muss der rvsjs-Dienst mit dem folgenden Befehl gestartet werden:

`rvsjs` oder

`rvsjs -b`

Gestoppt wird rvsjs mit dem folgenden Befehl:

`rvsjs -e`

**Hinweis**: rvsjs kann auch zusammen mit dem rvs® Monitor gestartet werden, indem in die Datei `$RVSPATH\init\rdmini.dat` die Anweisung

```
system cmd="rvsjs.exe –b"
```

aufgenommen wird.

**UNIX-Systeme**:

**Hinweis**: Es empfiehlt sich, zuerst die rvsjs Konfigurationsdatei `$RVSPATH\init\rvsjs.cnf` zu konfigurieren, da sie beim Start von rvsjs gelesen wird (sehe Kapitel 9.1.4).

Starten Sie rvsjs auf UNIX-Systemen mit dem folgenden Befehl:

```
rvsjs –t <Aktualisierungsintervall in Sekunden>
```

Diese Kommandozeile kann mit dem folgenden Befehl

```
system cmd="rvsjs -t <Aktualisierungsintervall in Sekunden>"
```

in die Datei `$RVSPATH/init/rdmini.dat` aufgenommen werden, da rvsjs selbständig terminiert, wenn sich der rvs® Monitor beendet und man infolgedessen bei einem Start des rvs® Monitors zusätzlich rvsjs starten müsste.

Beispiel:

```
system cmd="rvsjs -t 60"
```

Das Jobverzeichnis wird periodisch abgeprüft. Dieses Intervall ist mit der Option `-t` in Sekunden konfigurierbar (im obigen Beispiel sind das 60 Sekunden).

Beim Stopp vom rvs®-Monitor wird auch rvsjs gestoppt. Nur rvsjs stoppen, können Sie mit dem folgenden Befehl:

```
kill –s SIGQUIT <pid>
```

`<pid>` steht für die ProzessID des rvsjs-Prozesses.

### 9.1.6 Automatisches Versenden von Dateien mit Hilfe von rvsjs

Viele rvs®-Kunden verwenden rvsjs, um Dateien aus einem bestimmten Verzeichnis direkt zu versenden, ohne dabei eine `rvsbat`-Datei zu schreiben. In diesem Kapitel bringen wir einen Vorschlag, wie man rvsjs für diese Zwecke einsetzen kann. Dabei wird für den Dateiversand das rvs®-Tool `rvssce` angewendet.

In der rvsjs-Konfigurationsdatei `rvsjs.cnf` sollen die Parameter für Jobverzeichnis, Jobdateimuster und Jobapplikation gesetzt werden.

**Beispiel (Windows)**:

```
c:\out –g *kfg* sendfile.cmd
```

rvsjs wird das Verzeichnis `c:\out` nach Dateien, die dem Muster `*kfg*` entsprechen, durchsuchen und dann für jede die Jobapplikation `sendfile.cmd` ausführen.

In der Jobapplikation `sendfile.cmd` soll das rvs®-Tool `rvssce` aufgerufen werden. Dieses Tool dient zum Versenden der Dateien. Die Datei `sendfile.cmd` könnte folgendermaßen aussehen:

**Beispiel**:

```
rvssce -d %1 -s H78
```

Mit der Option `–d` wird der Dateiname der zu versendenden Datei übergeben (%1); mit der Option `–s` Stationsname der Station, an die die Datei versendet werden soll. Siehe bitte Kapitel 9.10 für die ausführliche Erklärung aller `rvssce`-Optionen.

### 9.1.7 Referenz

#### rvsjs in rvsXP

| | |
|---|---|
| rvsjs -? | Hilfetext und aktuelle Konfiguration anzeigen |
| rvsjs -p <Jobverzeichnis><br>[-a <Jobapplikation>]<br>[-j <Jobdateimaske>]<br>[-g] | Jobverzeichnis konfigurieren (wirksam nach Neustart); Jobsapplikation und Jobdateimaske sind optionale Parameter. Standard: `rvsbat` und `.job` (siehe Kapitel 9.1.4). |
| rvsjs -t <Aktualisierungsintervall in Sekunden> | Starten von rvsjs mit periodischer Überprüfung der Jobverzeichnisse. |
| rvsjs -r <Verzeichnis> | Verzeichnis aus Liste der Jobverzeichnisse entfernen (wirksam nach Neustart) |
| rvsjs -i | rvsjs als XP-Dienst installieren |
| rvsjs –i -s | rvsjs als XP-Dienst installieren, der mit Windows startet |
| rvsjs –i –u <Konto> -x <Kennwort> | rvsjs als XP-Dienst installieren, der ein bestimmtes Benutzerkonto nutzt |
| rvsjs -d | rvsjs als Windows-Dienst deinstallieren |
| rvsjs -b | rvsjs als Dienst starten |
| rvsjs -e | rvsjs stoppen |
| rvsjs -c | rvsjs in der Konsole starten |
| rvsjs -f | Normalerweise startet rvsjs nicht, wenn das mit der Option -p konfigurierte Verzeichnis nicht existiert (oder unerreichbar ist). Die Option `–f` erlaubt rvsjs trotzdem zu starten und rvsjs überprüft periodisch, ob das Jobverzeichnis schon vorhanden ist und sich öffnen lässt. |

#### rvsjs in rvs X

| | |
|---|---|
| rvsjs -? | Hilfetext anzeigen |
| rvsjs -p <Jobverzeichnis><br>[-a <Jobapplikation>]<br>[-j <Jobdateimaske>]<br>[-g] | Jobverzeichnis konfigurieren (wirksam nach Neustart); Jobsapplikation und Jobdateimaske sind optionale Parameter. Standard: `rvsbat` und `.job` (siehe Kapitel 9.1.4). |
| rvsjs -t <Aktualisierungsintervall in Sekunden> | Starten von rvsjs mit periodischer Überprüfung der Jobverzeichnisse. |
| rvsjs -f | Normalerweise startet rvsjs nicht, wenn das konfigurierte Jobverzeichnis nicht existiert (oder unerreichbar ist). Die Option `–f` erlaubt rvsjs trotzdem zu starten und rvsjs überprüft periodisch, ob das |

| | |
|---|---|
| | Jobsverzeichnis schon vorhanden ist und sich öffnen lässt. |
| rvsjs -c | Start rvsjs in der Konsole und nicht als Demon (Hintergrundprozess) |
| rvsjs –m <limit> | max. Anzahl aktiver KT-Kommandos |
| rvsjs –n <number> | max. Anzahl aktiver Jobs |

### 9.1.8 Fehlerwerte von rvsbat

| Situation | Fehlerwert |
|---|---|
| Versanddatei, die in Jobdatei aufgeführt ist, ist nicht vorhanden | 99 |
| kein Eintrag 'DSN' (Data Set Name, die Versanddatei) in der Jobdatei | 99 |
| kein Eintrag 'SID' (Ziel-ID) in der Jobdatei | 99 |
| falsche Syntax | 1 |
| falsche Syntax | 2 |
| falsche Syntax | 3 |

## 9.2 Sicherung der Stationstabelle (rvswrdstat)

Das Programm `rvswrdstat` für Windows- und Unix-Systeme ermöglicht Ihnen Ihre Stationsdaten aus der rvs® -Datenbank auf Bildschirm (stdout) herauszugeben oder in eine Datei zu sichern.

Die rvs®-Stationen werden bei Windows-Systemen über die graphische Oberfläche und auf Unix-Systemen (rvsX) in der Datei `$RVSPATH/init/rdstat.dat` gepflegt.

**Syntax:**

```
rvswrdstat [-?o]
```

Optionale Parameter:

**-?** Hilfe

**-o <Ausgabedatei>** schreibt die Stationsdaten in die Ausgabedatei; ohne diese Option werden die Daten auf die Standardausgabe (Bildschirm) geschrieben.

**Beispiel**:

```
rvswrdstat –o /home/skk/rvs/arcdir/rdstat12.dat
```

Die Sicherungsdatei kann mit den folgenden Schritten in rvs® importiert werden:

**Unix-Systeme**:

- Sichern Sie die alte Stationstabelle `$RVSPATH/init/rdstat.dat.`
- Kopieren Sie die Sicherungsdatei, die Sie mit Hilfe von `rvswrdstat` erzeugt haben (z.B. `rdstat12.dat`), ins Verzeichnis `$RVSPATH/init` und speichern sie unter dem Namen `rdstat.dat.`
- Damit die neuen Daten aus der Stationstabelle in rvs® auch aktualisiert werden, müssen Sie noch den Befehl `modst` in der Operator Konsole (`rvscns`) ausführen.
  **Beispiel:** `rvscns`
  `modst`

**Windows-Systeme:**

Auf den Windows-Systemen soll für den Import der Stationstabelle die graphische Benutzeroberfläche benutzt werden.

- Öffnen Sie mit dem Menübefehl `Ansicht` ⇒ `Stationen` im rvs®-Administrator das Fenster `Stationstabelle`.
- Führen Sie den Menübefehl `Bearbeiten` ⇒`Stationen exportieren` aus. Das Dialogfenster `File mit StationsDefinitionen auswählen` öffnet sich und zeigt die Dateien mit der Dateiendung `*.dat` im `arcdir` -Verzeichnis an.
- Wählen Sie die Datei aus, die die wiederherzustellende Stationstabelle enthält.
- Bestätigen Sie Ihre Auswahl mit **Öffnen**.

## 9.3 Konvertieren von U oder T Dateien zu Pseudo F oder V Dateien (`rvsut2fv`)

Auf vielen Plattformen, wie **PC** oder **UNIX**, gibt es keine speziellen Datenformate für die Dateien; alle Datensätze sind entweder ASCII oder binäre Dateien und werden deshalb entsprechend als Dateien im **T** (Text) oder **U** (unstructed=binär) Format von rvs® übermittelt.

Wenn Sie eine Datei zu einem anderen System schicken wollen, welches feste und/oder variable Datenformate unterstützt, und Sie sicher sein wollen, dass Ihre Daten mit einer bestimmten Datengröße geliefert werden, können Sie `rvsut2fv` benutzen, um Ihren Datenstrom in eine Datei mit (pseudo) festem/variablem Datenformat zu konvertieren, um sie dann mit der Angabe eines **F** (fixed) oder **V** (variable) Formats (über die Dialog- oder Kommandozeilen-Schnittstelle) zu senden. Diese Vorgehensweise ist z.B. notwendig, wenn Sie eine unstrukturierte Datei von

einem UNIX System zu einem MVS Host senden wollen, wo sie in einem bestimmten **F** oder **V** Format gespeichert werden muß. Weitere Informationen zum Senden von Dateien mit festen und/oder variablen Format ohne Verwendung von `rvsut2fv` sind im Abschnitt über rvs® Parameter (Parameter **VFTYP)** des Benutzerhandbuchs enthalten.

**Beachten** Sie, dass die automatische Code-Konvertierung zwischen ASCII und EBCDIC nur für Dateien im **T** Format stattfindet. Aus diesem Grund müßten Sie die Input- und Output-Codes spezifizieren, wenn Sie Dateien in einem anderen Format (**U**, **V**, **F**) versenden.

**Syntax**

```
rvsut2fv <output file> <F or V> <record length> <input file>
<U or T>
```

Alle Parameter sind erforderlich:

| | |
|---|---|
| **<output file>** | Der absolute Name der Output-Datei; das (Unter-) Verzeichnis muß existieren |
| **<F or V>** | Ein Buchstabe für Format der Output-Dateien (**F** oder **V**) |
| **<record length>** | Datenlänge für eine Output-Datei im **F** Format, maximale Datenlänge für eine Ouput-Datei im **V** Format |
| **<input file>** | Name der Input-Datei |
| **<U or T>** | Ein Buchstabe für das Format, das festlegt, ob die Input-Datei als eine binäre (**U**) oder eine Textdatei (**T**) zu interpretieren ist. |

**T**-Format Input-Dateien: Jede Zeile wird mit **Enter** und **Zeilenvorschub** begrenzt (carriage return + line feed). Jede Zeile wird in einen Output-Record konvertiert. Längere Zeilen werden auf **<record length>** gekürzt.

**F**-Format Output-Dateien: Kürzere Zeilen werden mit Leerzeichen ergänzt.

**U**-Format Input-Dateien: Jeder Outputrecord enthält <record length> Bytes, mit Ausname des letzten, das für **V** Format Output-Dateien kürzer sein kann.

**F**-Format Output-Dateien: Die Datei kann abschließende Nullen enthalten.

Enthält eine Datei Leerzeichen, ergänzt `rvsut2fv` diese Stelle mit einem Leerzeichen. Im Format **Variable** wird diese Leerzeile als Satz übertragen, der genau ein Byte enthält. Im Format **Fixed** wird dieser Satz bis zur Satzlänge mit Leerzeichen aufgefüllt.

## 9.4 Aktive Panel (`rvsap`)

Durch dieses rvs® Werkzeug ist der Administrator in der Lage, mehr Informationen über den Status und den Fortschritt der Übertragung zu bekommen. Es ist nur für Windows, AIX, IRIX und Solaris verfügbar.

Dieses Programm können Sie starten, indem Sie

- das Kommando `rvsap` aufrufen (für UNIX Systeme)
- den Menübefehl `Ansicht` → `Aktive Leitungen` im rvsNT/rvsXP-Administrator aufrufen (für Windows)

Folgende Details stehen im Panel zur Verfügung:

| | |
|---|---|
| **SID** | Stations-ID |
| **State** | Status der Übertragung auf der Ebene des Odette Protokolls |
| **R(R)** | Ich empfange eine Datei, und ich bin der Initiator (die aktive Seite) des Kommunikationsprozesses. |
| **S(S)** | Ich sende eine Datei, und ich bin der Initiator (die aktive Seite) des Kommunikationsprozesses. |
| **R(S)** | Ich empfange eine Datei, und ich bin der Reagierende (die passive Seite) des Kommunikationsprozesses. |
| **S(R)** | Ich sende eine Datei, und ich bin der Responder (die passive Seite) des Kommunikationsprozesses. |
| **Lyne Type** | Kommunikationstyp |
| **Process ID** | ID des Kommunikationsprozesses auf der Ebene des Betriebssystems |
| **DSN** | Name der Datei, die gerade übertragen wird. |
| **v(B/s)** | Übertragungsgeschwindigkeit |
| **rate** | Anteil der Übertragung (prozentual), der schon gesendet worden ist. |
| **start** | Uhrzeit des Übertragungsbeginns |

Wenn Sie gleichzeitig viele besetzte Leitungen haben, aber Ihr Interesse nur auf ein paar von ihnen gerichtet ist, können Sie ein Filter definieren, das Ihnen nur die Objekte Ihres Interesses anzeigt. Sie können das Filter über die folgenden Parameter definieren:

- eine **SID** beschränkt die Auswahl auf Leitungen, die zu dieser Station führen, * bedeutet keine Stationsbeschränkung
- ein Kommunikationstyp **TYPE** beschränkt die Auswahl der Leitungen für diesen Kommunikationstyp, * bedeutet keine Kommunikationstyp-Beschränkung

Die Funktionstaste **<F3>** oder die **<ESC>**-Taste schließen den Panel auf UNIX Systemen.

### 9.5 Wiederherstellen des Isam Index (`rvsrii`)

Die rvs® Datenbank ist nach der isam Methode (index sequential access method) organisiert.

Im `$RVSPATH/db/` Verzeichnis gibt es zwei Dateitypen:

`*.db`

`*.idx`

Die `*.db` Dateien sind Tabellendateien, und die `*.idx` Dateien sind Indexdateien.

Jeder Zugang zu einer Tabellendatei läuft über eine `*.idx` Datei. Indexdateien können sehr groß werden, deshalb müssen sie regelmäßig erneuert werden.

```
rvsrii ['/eenvdsn'] ['/lx']
```

Alle Parameter sind optional:

- der optionale Parameter **/e** wird nur benutzt, wenn die Umgebungsdatei nicht in der Umgebungsvariable **RVSENV** definiert ist und sich auch nicht im aktuellen Verzeichnis befindet;
- der optionale Parameter **/l** definiert die Sprache (**x**), die für Eingabeaufforderungen und Nachrichten benutzt werden soll. Der Standard ist Englisch.

**Beispiel**:

```
rvsrii /ld
```

Mit diesem Kommando können Sie `rvsii` aufrufen, mit Deutsch als Sprache für Eingabeaufforderungen und Meldungen.

Diese Funktion kann gestartet werden, während rvs® im Betrieb ist.

Sie sollten dieses Kommando aus der Datei

`$RVSPATH/init/rdmini.dat`

aufrufen, indem Sie das Kommando `opcmd` eingeben:

```
OPCMD cmd="system cmd='nohup rvsrii > /dev/null &'"
time=02:00:00 repeat=24:00:00
```

Dieses Kommando bewirkt, dass eine Datenbank-Indexbereinigung jeden Tag um 2 Uhr stattfindet.

Wenn Sie gezwungen sind, das `rvskill` Kommando zu benutzen, ist es dringend notwendig, danach die `rvsrii` Funktion anzuwenden, um mögliche Beschädigungen der rvs® Datenbank zu umgehen.

## 9.6 rvs® Informationseintrag (`rvsie`)

`rvsie` (das rvs® Informationseintrag-Recovery-Tool), kann sehr hilfreich sein, wenn Ihre Datenbank beschädigt wird. `rvsie` macht es Ihnen möglich, alle Informationseinträge wiederherzustellen, die noch nicht erfolgreich beendet wurden, da genau solche unvollendeten rvs® Kommandos verloren gehen können, wenn die Datenbank beschädigt wird.

Welche Informationseintragsarten können durch `rvsie` wiederhergestellt werden?

- Alle `SE`s von Dateien, die nicht vollständig übertragen wurden
- Alle `IE`s von Dateien, die vollständig empfangen wurden, aber zum Routing bestimmt sind
- Alle `QS`s, die noch zu senden sind

**Syntax:**

```
rvsie [-fvsdriomtqanlc]
```

**-f** Lokaler Dateiname (**DSNLOCAL**)

**-v** Virtueller Dateiname (**VDSN**)

**-s** SID des Senders (**SIDSENDER**)

**-d** SID des Ziels (**SIDDEST**)

**-r** Datenformat (**RECFM**)

**-i** Eingabe-Code (**A**=ASCII oder **E**=EBCDIC)

**-o** Ausgabe-Code (**A**=ASCII oder **E**=EBCDIC)

**-m** Maximale Datenrecordlänge (**MAXRECL**)

**-t** Zeit (**DTAVAIL**)

**-q** Generiere eine End-to-End-Response (**EERP**)

**-a** unverzüglicher Anruf zum Partner (nur **EERP** Eigenschaft)

**-n** Optionale Nachbar-SID (nur **EERP** Eigenschaft)

**-l** schreibt Liste der Einträge in die rvs® Datenbank (Input für

`rvsie`)

**-c** schreibt Liste der Einträge in der rvs® Datenbank (Input für `rvsie`)

Um Ihre beschädigte Datenbank durch `rvsie` wiederherzustellen, sollten Sie die folgenden Schritte befolgen:

- Mit dem Kommando
  `rvsie –l`
  können Sie alle Informationseinträge aus der Datenbank auflisten, die noch nicht ausgeführt wurden. Für jeden Informationseintrag wird ein entsprechender Aufruf des `rvsie` Kommandos generiert.
- Der nächste Schritt ist, diesen Output in eine Kommandodatei umzuleiten:
  Beispiel für **Windows**: `rvsie -l > restore.bat`
  Beispiel für **UNIX**: `rvsie -l > restore`
- Löschen Sie jetzt die beschädigte Datenbank mit dem Kommando:
  `rvsdbdel`
- und erstellen Sie eine neue mit dem Kommando:
  `rvsidb lid`
  wobei **lid** Ihre lokale Stations-ID ist.
- Führen Sie danach die neu generierte Kommandodatei aus:
  Beispiel für **Windows**: `restore.bat`
  Beispiel für **UNIX**: `sh restore`.

Wenn es in der Ausgabedatei Informationseinträge gibt, die für Sie nicht mehr wichtig sind, bearbeiten Sie diese Datei vor ihrer Ausführung und löschen Sie alle irrelevanten Informationseinträge. Bei der Ausführung der Ausgabedatei wird das Kommando `rvsie` für jeden nicht ausgeführten Informationseintrag automatisch aufgerufen. Sie brauchen es nicht manuell für jeden Informationseintrag tun.

## 9.7 Backup der rvs® Daten (`rvsbackup`)

Es ist sehr wichtig, regelmäßig ein Backup der rvs® Daten zu erstellen, um einem eventuellen Datenverlust, z.B. durch Stromausfall, vorzubeugen. Dieses Tool hilft Ihnen diese Aufgabe automatisch zu erfüllen.

`rvsbackup` überprüft, dass

- kein anderer rvs® Prozeß läuft, und
- das ausgewählte Archiv-Verzeichnis existiert.

`rvsbackup` erstellt

- einen Dump der rvs® Datenbank (`rvsdbdump.log`)

- eine rvsenv.var Datei, die den Wert von **$RVSENV** enthält (**Beispiel**: $RVSPATH/rvsenv.dat)

rvsbackup erstellt Kopien von

- allen Datenbank Log-Dateien (rlog.log, rlstat.log, rldb.log)
- allen Dateien aus dem $RVSPATH/temp Verzeichnis
- der Konfigurationsdatei ($RVSPATH/rvsenv.dat)
- allen Dateien aus dem $RVSPATH/init Verzeichnis (rdkey.dat, rdmini.dat, rdstat.dat).

**Syntax:**

```
rvsbackup [optionen]
```

| | |
|---|---|
| **-?** | Syntax der Benutzung |
| **-a** | Alle Backupschritte in $ARCDIR |
| **-e** | Speichern von $RVSENV |
| **-b** | Dump der Datenbank |
| **-x** | Löschen des Datenbank-Logs nach erfolgreichem Dump |
| **-s** | Überprüfen, ob rvs® gestoppt ist. |
| **-c** | Backup-Kopie (rlog.log, rldb.log, rlstat.log, init/*, rvsenv.dat, temp/*) |
| **-r** | Aufrufen von prervsbackupext |
| **-o** | Aufrufen von postrvsbackupext |
| **-n** | Erstellen und Benutzen eines neuen Unterverzeichnisses in $ARCDIR |
| **-d dir** | Speichern im Verzeichnis dir |

**Beispiele:**

```
rvsbackup -a
```

Dieses Kommando führt **alle** Backup-Schritte in das Archiv-Verzeichnis $ARCDIR aus.

```
rvsbackup -a -d /home/tmp/backup
```

Dieses Kommando führt **alle** Backup-Schritte in das Verzeichnis `/home/tmp/backup aus`.

Es besteht die Möglichkeit, den Backup-Prozeß mit Ihren eigenen installationsspezifischen Shell Scripts zu erweitern. Das Shell Script `prervsbackupext`, wenn es existiert, wird nach allen Überprüfungen und vor dem Zugriff auf die rvs® Datenbank aufgerufen. Das Shell Script `postrvsbackupext`, wenn es existiert, wird aufgerufen, nachdem alle Backup-Schritte getan sind. Die beiden Shell Scripts werden im selben Verzeichnis erwartet.

Wenn Sie mit Hilfe dieses Tools ein Backup erstellt haben, können Sie den gespeicherten Zustand des rvs® Systems mit dem Tool `rvsrestore` wiederherstellen.

## 9.8 Wiederherstellung der rvs® Daten (`rvsrestore`)

Dieses Tool dient dazu, den vorherigen Zustand des rvs® Systems aus dem Backup wiederherzustellen, das mit Hilfe des `rvsbackup` Tools erstellt wurde.

`rvsrestore` stellt sicher, dass

- kein anderer rvs® Prozeß läuft.

`rvsrestore` kopiert das Backup

- rvs® Umgebungsdatei (`$RVSPATH/rvsenv.dat`)
- alle Dateien aus dem `$RVSPATH/init` Verzeichnis (`rdkey.dat`, `rdstat.dat`, `rdmini.dat`)
- rvs® Datenbank Log-Dateien (`rlog.log`, `rlstat.log`, `rldb.log`)
- alle Dateien aus dem `$RVSPATH/temp` Verzeichnis

`rvsrestore` löscht die alte Datenbank, erstellt eine neue und füllt sie mit den alten Daten aus dem Backup.

**Syntax**

```
rvsrestore [optionen]
```

| | |
|---|---|
| **-?** | Syntax der Benutzung |
| **-a** | Alle Wiederherstellungsschritte |

| | |
|---|---|
| **-s** | Überprüfen, ob rvs® gestoppt ist. |
| **-e** | Kopieren von $**RVSENV** (`rvsenv.dat`) |
| **-k** | Kopieren von `rdkey.dat` |
| **-r** | Kopieren von `rdstat.dat` |
| **-i** | Kopieren von `rdmini.dat` |
| **-l** | Kopieren von `rlog.log` |
| **-o** | Kopieren von `rlstat.log` |
| **-d** | Wiederherstellen der Datenbank aus der Dumpdatei |
| **-m** | Kopieren der Dateien aus dem `temp` Verzeichnis |
| **-b** | Wiederherstellen der Datenbank aus `rldb.log` |
| **-x** | Löschen von `rldb.log` nach erfolgreicher Wiederherstellung der Datenbank |
| **-f dir** | (aus dem) Archivverzeichnis |

**Beispiel**:

```
rvsrestore –a –f /home/skk/rvs/arcdir
```

Dieses Kommando führt alle Wiederherstellungsschritte aus dem Archivverzeichnis (`/home/skk/rvs/arcdir`) aus.

## 9.9 rvs® End-to-End Response (rvseerp)

**EERP** (End-to-End Response) ist ein wichtiger Dienst des Odette Protokolls. Er soll vom Endempfänger an den ursprünglichen Absender der Originaldatei gesendet werden. Eine Datei gilt nur dann als komplett geliefert, wenn auch der **EERP** empfangen worden ist. Es kann passieren, dass Kommandos in Ihrer rvs® Datenbank nicht beendet werden, wenn Ihr Partner keinen **EERP** geschickt hat, oder wenn Sie nicht imstande sind einen **EERP** an Ihren Partner zu senden.

Das Tool `rvseerp` ermöglicht es Ihnen,

1 Sendeprozesse, die noch ohne **EERP** sind,

2 Empfangsprozesse, für die kein **EERP** gesendet werden konnte

anzuzeigen.

Im ersten Fall bleibt das `SK` im "pending" Status. Dasselbe gilt auch für das `QK` im zweiten Fall. Weitere Informationen entnehmen Sie Kapitel 2.1.1"rvs® Monitor - Basiseigenschaften".

**Syntax**

```
rvseerp [-?lceqsnt]
```

Dieses Programm hat drei Grundfunktionen:

1. Liste "pending" SK/QS Kommandoeinträge (Option **–l**) auf
2. Beende "pending" SK Kommandoeinträge (Option **–c**)
3. Beende "pending" QK Kommandoeinträge (Option **–e**)

Der Output der ersten Funktion (Option **-l**) enthält ausführliche Information über den Kommandoeintrag im "pending" Status und das `rvseerp` Kommando, das zum Beenden dieses Eintrags verwendet werden kann. Das `rvseerp` Kommando wurde als Kommentar geschrieben (rem für **NT**, # für **UNIX**).

Die Zusatzfunktionen für die **-l** Option sind:

- -q
- -s
- -t

**Beispiel**:

```
rvseerp –l –s –t 24
```

listet alle `pending` SKs auf, die älter als 24 Stunden sind.

Sie können die Ausgabe von `rvseerp –l` in eine Datei umleiten und ihn bearbeiten, indem Sie alle Kommentarzeichen der Kommandos entfernen. `rvseerp` wird durch die Optionen `–c` oder `–e` (die zweite und die dritte Grundfunktion von `rvseerp`) aufgerufen, um entsprechend alle `pending` SKs und QSs zu beenden.

**Beipiel**:

```
rvseerp –l
```

zeigt eine ausführliche Liste der `pending` Kommandoeinträge mit ihren Kommandonummern.

`rvseerp –l > output.bat` (für **NT**)

`rvseerp –l > output` (für **UNIX**)

leitet den Output von `rvseerp –l` in eine Output-Datei um.

Jetzt können Sie die Output-Datei bearbeiten und alle Kommentarzeichen der Kommandos entfernen, die Sie ausführen möchten. Führen Sie anschließend die bearbeitete Datei aus.

`output.bat` (für **NT, XP**)

`sh output` (für **UNIX**)

Die zweite (`-c`) und die dritte (`-e`) Grundfunktion können getrennt ausgeführt werden:

**Beispiele**:

`rvseerp –c –n XXXXXX` (für `SKs`)

`rvseerp –e –n XXXXXX` (für `QSs`)

`XXXXXX` ist die Kommandonummer aus der rvs® Datenbank, die durch `rvseerp –l` generiert wurde.

Sie sollten dieses Tool sehr vorsichtig und selten benutzen, weil es das Standard-Kommunikationsprotokoll (**Odette**) umgeht. Die bessere Lösung ist es, die Odette Parameter **EERP_IN** oder **EERP_OUT** in Ihrer Stationtabelle zu benutzen.

**Hinweis**: Mit dem Programm `rvseerp` ist es auch möglich, EERPs zu löschen oder freizugeben. Diese Funktion ist im Benutzerhandbuch rvsX, Kapitel 3.1.7 beschrieben. In rvsxP können Sie die EERPs über die graphische Oberfläche freigeben oder löschen (siehe Benutzerhandbuch, Kapitel 6.3.4).

## 9.10 rvssce

Das Hilfswerkzeug `rvssce` bietet Ihnen ab der Version 2.05 für rvsX, rvsNT und rvsXP eine zusätzliche Möglichkeit Datei zum Partner zu versenden. Alle Versandparameter, die Ihnen in `rvsdia` (rvsX) oder rvsNT/rvsXP, `rvsbat` und `rvscal` zur Verfügung stehen, sind hier mit den gleichen Ausprägungen möglich.

Mit `rvssce` können Sie auch den Status eines Kommandos aus der rvs® Datenbank abfragen und diese Information in eine XML-Datei umlenken, um sie eventuell weiterzuverarbeiten.

## Dateiversand

### Syntax:

```
rvssce -d <Dateiname> -s <StationsID>

[-uvtlIoODFTSVMYCfi]
```

### Obligatorische Parameter

**-d** Name der zu versendenden Datei.

**-s** SID der Zielstation.

### Optionale Parameter

**-u** Lokale Benutzer-ID.

**-v** virtueller Dateiname für die Übertragung (VDSN), max. 26 Zeichen lang.

**-V** Textformat der zu sendenden Datei (siehe Referenzhandbuch, Kapitel 10.6 oder Benutzerhandbuch, Kapitel “Die rvs® Parameter im Überblick“, Parameter **VFTYP**).

**-D** Angabe, ob die Datei nach dem erfolgreichen Versenden gelöscht oder erhalten bleiben soll:

Wählen Sie:

- **K**, um die Datei nach dem Senden beizubehalten
- **D**, um die Datei nach dem Senden zu löschen.

Standard: **K**

**-l** Label (max. 20 Zeichen lang); Name der Gruppe der serialisierten Sendeaufträge.

Benutzerspezifische (beschreibende) Kennung für eine Gruppe der Sendeeinträge. Es wird bei **Serialisieren=Y** für das Serialisieren mit einem anderen Sendeeintrag mit derselben Kennung verwendet. Dadurch wird gewährleistet, dass die Dateien in einer geordneten Reihenfolge ankommen.

**-F** Format der zu versendenden Datei.

Wählen Sie:

- **T** = Textdatei; eine Folge von ASCII-Zeichen
- **U** = unstrukturierter (binärer) Datei
- **V** = variable Satzlänge
- **F** = feste Satzlänge
- **leer** = Standard des Systemformtes

(d.h. **U** für rvsNT/rvsXP und rvsX, **F** für rvs400)

**-T** Versandzeit; gibt den Zeitpunkt an, zu dem die Datei frühestens gesendet wird.

Wählen Sie:

- **H** = Datei ist im Status angehalten (held); er wird erst gesendet, wenn Sie oder der rvs® Administrator ihn freigeben.
- **explizite Zeit** = Jahr/Monat/Tag Stunde:Minute (**JJJJ/MM/TT HH:MM:SS**)
- **leer** = jetzt

Standard: **leer** = jetzt

**-S** Serialisierung; geben Sie an, ob diese Datei zu einer Gruppe von serialisierten Dateien gehört. (siehe das Feld `Kennung/Label`)

Wählen Sie:

- **Y**(es) für serialisiertes Senden der Datei
- **N**(o) für nicht serialisiertes Senden der Datei

Standard: **N**

**-I** Eingabe-Code (**A**=ASCII oder **E**=EBCDIC).

**-O** Ausgabe-Code (**A**=ASCII oder **E**=EBCDIC).

**-t** Pfad der eigenen Umwandlungstabelle.

**-M** Satzlänge, hat die gleiche Bedeutung wir der Parameter **MAXRECL** (siehe Kapitel 10.6).

**-Y** Verschlüsselung; Wählen Sie :

- **Y** = Ja
- **N** = Nein.

**-C** Komprimierung; ; Wählen Sie :

- **Y** = Ja
- **N** = Nein.

**-o** Umlenkung der Ausgabe in XML-Datei.

**-f** Odette-ID des Senders (Originator)

**-i** Odette-ID der Zielstation

**Beispiel**:

```
rvssce -d /home/skk/out/test2.txt -s D43
```

Die Datei `/home/skk/out/test2.txt` wird zur Station `D43` versandt.

Sie haben auch die Möglichkeit den Befehl `rvssce` in die Datei `$RVSPATH/init/rdmini.dat` einzutragen, um eine Datei regelmäßig in bestimmten Zeitabständen zu versenden.

**Beispiel:**

```
opcmd CMD="system CMD='rvssce -d D:\Testdaten\test42.txt -s DL3'" repeat=04:00:00
```

In diesem Beispiel wird die Datei `test42.txt` alle 4 Stunden an die Station `DL3` versendet.

## Abfrage des Status eines rvs® -Kommandos:

**Syntax:**

```
rvssce -c <Kommandonummer> -o <XML-Datei >
```

**Beispiel**:

```
rvssce -c 89 -o /home/skk/out/ausgabe.xml
```

Hier wird der Status des Kommandos Nummer `89` in die Datei `/home/skk/out/ausgabe.xml` ausgegeben. Der Status des Kommandos `89` ist z.B. beendet (`<ended>`).

**Hinweis:** Allerdings muss der Monitorparameter **SSCREATE** (siehe Benutzerhandbuch, Kapitel "Die rvs® Parameter im Überblick", Parameter **SSCREATE**) gleich 1 sein. Nur in diesem Fall werden die Informationen über die erfolgreich abgeschlossenen Kommandos in der Statistik-Datenbanktabelle **SS** weitergeführt. Dies verursacht keine Performanzprobleme mit der rvs® Datenbank.

`rvssce` gibt im Erfolgsfall 0 zurück. Im Fehlerfall werden u.a. folgende Fehlercodes zurückgegeben:

```
#define RC_INVALID_ARG                  11
#define ERROR_LOAD_LIBRARY              12
#define ERROR_GETPROCESSADDRESS         13
#define ERROR_CANNNOT_OPEN_XMLOUT       14
#define ERROR_CANNOT_CREATE_SE          15
```

### 9.11 Senden einer Datei (`rvssend`) nur für UNIX

`rvssend` läßt Sie eine Testdatei an eine rvs® Station senden:

```
rvssend local_filename remote_filename sid
```

Dieses Kommando ist ein einfaches Shell-Skript im Systemverzeichnis. Es kann zum Testen einer Verbindung benutzt werden. Das folgende Beispiel sendet das Systemprofil als "HELLO" an die Station "ABC".

```
rvssend /etc/profile HELLO ABC
```

## 9.12 rvscheckdb

Das Programm `rvscheckdb` analysiert die Tabellen der rvs®-Datenbank auf mögliche Inkonsistenzen und kann, wenn nötig, diese beseitigen. Inkonsistenzen können unter anderem dann auftreten, wenn der Dateiversand/-empfang plötzlich unterbrochen wurde, z.B. im Falle einer fehlerhaften Kommunikationsverbindung.

**Syntax:**

```
rvscheckdb [-?| -o <Dateiname>| -T <Tabellenname>
-C <Kommandonummer> -d]
```

Optionen:

| | |
|---|---|
| **-?** | Hilfe |
| **-o <Dateiname>** | Ausgabedatei: das Ergebnis des Programms `rvscheckdb` können Sie in eine Ausgabedatei umleiten. |
| **-T <Tabellenname>** | Tabelle der rvs® -Datenbank, aus welcher die Kommandos gelöscht werden sollen. |
| **-C <Kommandonummer>** | Nummer des Kommandos, das gelöscht werden soll. |
| **-d** | Angabe der Aktion, die mit einem Kommando durchgeführt werden soll: d (delete) bedeutet Löschen |

**Benutzung:**

In der Kommandozeile des jeweiligen Betriebssystems den folgenden Befehl eingeben:

```
rvscheckdb -o <Dateiname>
```

**Beispiel** (Windows):

```
rvscheckdb –o C:\dbausgabe
```

Das Ergebnis dieses Beispiels ist eine Zusammenfassung der Datenbankprüfung, die in der Kommandozeile des Betriebssystems ausgegeben wird.

Darüber hinaus generiert `rvscheckdb` zwei text-Dateien, in welchen die Ergebnisse der Datenbankprüfung in ausführlicherer Form dargestellt werden:

- `<Dateiname>.ok` : mit einer Auflistung aller konsistenten DB-Einträge,

- `<Dateiname>.err`: mit einer Auflistung aller inkonsistenten DB-Einträge.

und ein Löschskript. In Abhängigkeit vom verwendeten Betriebssystem findet sich dieses unter:

- `<Dateiname>.sh` für Unix-Betriebssysteme, oder
- `<Dateiname>.bat` für Windows-Betriebssysteme.

**Beispiel:**

C:\WINDOWS\System32\cmd.exe

```
Zusammenfassung:
----------------

Anzahl gepruefter SE-Eintraege aus ET:      2
        davon unvollstaendig:               0

Anzahl gepruefter IE-Eintraege aus ET:      0
        davon unvollstaendig:               0

Anzahl gepruefter VD-Eintraege:             2
        davon unvollstaendig:               0

Anzahl gepruefter SE-Eintraege:             2
        davon unvollstaendig:               0

Anzahl gepruefter SK-Eintraege:             2
        davon unvollstaendig:               0

Anzahl gepruefter IE-Eintraege:             1
        davon unvollstaendig:               0

Anzahl gepruefter IZ-Eintraege:             1
        davon unvollstaendig:               0

Anzahl gepruefter QS-Eintraege:             1
        davon unvollstaendig:               1

Detailiertere Informationen finden Sie in den log-Dateien dbausgabe.ok und dbaus
gabe.err .
Die Datei dbausgabe.bat enthaelt die Kommandos zum Bereinigen der Datenbank.
Sie kann als Skript ausgefuehrt werden. Vor dem Ausfuehren sind die
Kommentarzeichen vor den zu loeschenden Eintraegen zu entfernen.
```

Mit dem obigen Befehl `rvscheckdb -o C:\dbausgabe` wurden folgende Dateien erzeugt:

`dbausgabe.ok`, `dbausgabe.err` und `dbausgabe.bat`.

In der Folge hat der Anwender nun die Möglichkeit, das Löschskript mittels eines Editors seinen Wünschen anzupassen. Unter „Anpassung" wird dabei das Entfernen der jeweiligen Kommentarzeichen vor den Einträgen verstanden, die der Benutzer aus der Datenbank entfernen möchte.

Auf Unix-Betriebssystemen ist vor der Ausführung des Skripts das Kommentarzeichen „#" vor zu löschenden Einträgen zu entfernen.

Im Falle eines Windows-Betriebssystems ist die Zeichenfolge „REM echo" vor den gewünschten Löschbefehlen zu entfernen.

Das nachfolgende Beispiel zeigt die Darstellung eines fehlerhaften Sendjob-Eintrages innerhalb des Löschskripts.

**Beispiel**: dbausgabe.bat

```
REM echo Sendjob 1400764 inkonsistent!
REM echo <UNKNOWN> sollte um 0 von RVSFARM an LIN4 als
REM echo LOOP.TENNIS.U000.QPRQJQ versendet werden.
REM echo zu loeschende DB-Eintraege:
REM echo SE 1400764 KT 1400764 ET 1400764

REM echo rvscheckdb -T SE -C 1400764 -d
REM echo rvscheckdb -T KT -C 1400764 -d
REM echo rvscheckdb -T ET -C 1400764 -d
```

Die oberen Zeilen im Beispiel dienen der näheren Erklärung. Bei ihnen ist das Kommentarzeichen am Zeilenanfang (REM echo) **NICHT** zu entfernen.

In der zweiten und der dritten Zeile wird mittels der vorhandenen Job-Informationen ein Überblick über den Sende-/Empfangsauftrag gegeben. Die Darstellung arbeitet dabei nach folgendem Muster:

<Dateiname> sollte um <Uhrzeit> von <Sender> an <Empfänger> als <virtueller Dateiname> versendet werden.

Lassen sich einzelne Informationen zu dem Auftrag nicht aus der Datenbank entnehmen, so wird an diese Stelle <UNKNOWN>, bzw. 0 (für die Uhrzeitangabe) als Default-Angabe gesetzt.

Die letzen drei auskommentierten Zeilen des obigen Beispiels enthalten die eigentlichen Löschbefehle, welche das Programm rvscheckdb interpretieren und ausführen kann.

**Hinweis**: Je nach Vollständigkeit kann die Anzahl der Löschkommandos für einen einzelnen Eintrag variieren. Nur für diese Zeilen ist bei Bedarf das Kommentarzeichen zu entfernen!

Nachdem das Löschskript durch den Anwender editiert und Änderungen gespeichert wurden, lässt es sich abschließend in der Kommandozeile des Betriebssystems ausführen.

### 9.13 Datei verschieben (rvsmove)

Dieses Programm verschiebt eine Datei in ein anderes Verzeichnis und bietet gleichzeitig die Möglichkeit dieselbe Datei nach gewünschten

Dateinamenkonventionen umzubennen. Die Umbennenungsfunktionalität nach bestimmten Mustern ist nicht so einfach mit den Standard-Möglichkeiten, die Windows-Systeme anbieten, realisierbar, sodass sich viele Kunden dies in rvs® als Prozessunterstützung gewüscht haben.

Syntax:

```
rvsmove /src:<Quelle> /dst:<Ziel> /retry:<Versuche> /B /T /S:<Sufffix>
/D:<Delimiter>
```

Wenn Sie als Parameter nur Quelle und Ziel benötigen, ist folgende einfachere Syntax anzuwenden.

```
rvsmove <Quelle> <Ziel>
```

**Obligatorische Parameter**

| | |
|---|---|
| **/src:<Quelle>** | Name der zu verschiebende Datei. |
| oder | |
| **<Quelle>** | |
| **/dst:<Ziel>** oder | Das Verzeichnis und/oder Dateinamen der neuen Datei(en) |
| **<Ziel>** | |
| **/B** | Als neuer Name der Datei wird der Basisname (urspünglicher Name, der bei der Option `/src:` angegeben ist) genommen.<br>Hinweis: Dieser Parameter ist nur bei der längeren Syntaxform vorhanden. |

**Optionale Parameter**

| | |
|---|---|
| **/retry:** | Anzahl wie oft im Fehlerfall `rvsmove` versuchen soll eine Datei zu verschieben.<br>Standard: 10 |
| **/T** | Ein Timstamp in der Form YYYYDDMMhhmmssnnn wird am Anfang der Datei gesetzt<br>Beispiel:<br>`20061018112142091sendjob.bat`<br>Vor dem Dateinamen `sendjob.bat` wird der Timestamp `20061018112142091` gesetzt. |
| **/S:<suffix>** | Hänge ein Suffix an den neuen Dateinamen |

| | |
|---|---|
| | **Beispiel**: `/S: ht6` |
| | Ergebnis ist z.B. `sendjob.bathf6`. |
| **/D:<delimiter>** | Trennzeichen zwischen Elementen des neuen Namens |
| | Beispiel: /D:. |
| | Ergebnis: `20061018112142091.sendjob.bat.hf6` |

**Beispiele**:

```
rvsmove C:/text.txt D:/temp
```

Dieser einfache Befehl verschiebt die Datei `C:/text.txt` ins Verzeichnis `D:/temp`.

```
rvsmove /src:help.txt /dst:D:/temp /retry:15 /B /T /S:src /D:.
```

**Ergebnis**: Die Datei `help.txt` aus dem aktuellen Verzeichnis wurde ins Verzeichnis `D:/temp` verschoben und nach `20061118112142093.help.txt.src`. umbenannt. Der Dateiname besteht aus einem Zeitstempel am Anfang und einem Suffix am Ende. Die Elemente wurden mit einem Punkt getrennt. Im Fehlerfall wird 15-Mal versucht die Datei zu verschieben (`retry:15`).

### 9.14 rvshalt

Dieses Tool (Shell-Skript) stoppt rvsX und wartet bis alle Prozesse terminiert sind.

**Syntax:**

```
rvshalt -?
```

**Usage**:

```
rvshalt
```

# IV. rvs® Schnittstellen

In diesem Teil werden die beiden Schnittstellen rvs® Kommandozeilen-Schnittstelle (`rvsbat`) und C-CAL-Schnittstelle (`rvscal`) beschrieben.

## 10 rvs® Kommandozeilen-Schnittstelle und C-CAL-Schnittstelle

Dieses Kapitel beschreibt, wie Sie die beiden Schnittstellen starten und welche Parameter Sie verwenden können. Außerdem erhalten Sie eine Beschreibung der korrespondierenden globalen Kommandos von rvs®. Die Syntax der Kommandos wird ebenso erklärt wie der Prototyp der Funktion `rvscal()`.

Für alle Aufgaben (Senden und Empfangen von Dateien ebenso wie die Administration von rvs®), die Sie mit rvs® zu erledigen haben, können Sie entweder

- Kommandos in eine Textdatei schreiben und an die Kommandozeilen-Schnittstelle (`rvsbat`) übergeben
- Kommandos an die Funktion `rvscal()` der C-Cal-Schnittstelle übergeben
- C-Funktionen für die C-Cal-Schnittstelle benutzen (für jedes Kommando eine eigene Funktion)

Die Kommandos von `rvscal()` (C-Cal-Schnittstelle) sind identisch mit denen von `rvsbat` (Kommandozeilen-Schnittstelle) und werden im Kapitel 10 erläutert.

Die C-Funktionen der C-Cal Schnittstelle sind im Kapitel "Arbeiten mit C-Funktionen" beschrieben.

### 10.1 Die rvs® Kommandozeilen-Schnittstelle (rvsbat) starten

Die Kommandozeilen-Schnittstelle kann aufgerufen werden durch:

```
rvsbat [/c] [/e<envdsn>] [/i<cmdfile>] [/l<language>] [/q]
```

**Kommandozeilen-Parameter**

- **/c**: nach einem Fehler während der Ausführung des Kommandos rvsbat fortsetzten. Im Standard beendet `rvsbat` nach einem Fehler die Ausführung.

- **/e**: vom Standard abweichende Umgebungsdatei `$RVSPATH/rvsenv.dat` verwenden.

- /**i**cmdfile: Kommandos werden nicht von `stdin` gelesen, sondern aus der Kommando-Eingabedatei `cmdfile`. Die Kommando-Eingabedatei enthält folgende Elemente:
  - Kommandos (darf mehrere Zeilen lang sein; am Ende jeder fortzusetzenden Zeile muss das **+** stehen)
  - Kommentarzeilen (beginnend mit *)

- **/l**`language`: Nachrichtensprache verwenden, wie in `language` angegeben.

- **/q** Kommando ohne Echo (Stumm-Modus) ausführen; Rückmeldungen über Erfolg oder Fehler der Operation werden weiterhin unterstützt.

**Kommandobehandlung**

Kommandos können in die Kommando-Eingabedatei oder mit dem Aufruf von `rvsbat` geschrieben werden.

In beiden Fällen ist die Syntax des Kommandos identisch.

**Beispiel**:

```
SEND /C DSN=c:\temp\readme.txt (SID=LOC DSNNEW=new.txt)
```

Eine genaue Beschreibung der Kommandos ist in Kapitel 10 enthalten.

Für jedes von der Kommandoeingabe gelesene Kommando wird eine Nachricht zu **stdout** geschrieben, die Informationen über den Erfolg oder Fehler bei der Bearbeitung des Kommandos enthält.

Wenn `rvsbat` ohne Eingabe einer Kommandodatei gestartet wird, kann es mit **<STRG> C** abgebrochen werden.

**Beispiele:**

Die folgende Abbildung zeigt Ihnen ein Beispiel für das Senden einer Datei mittels `rvsbat` unter **Windows NT**:

```
Eingabeaufforderung
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

c:\rvssrc>rvsbat
rvsNT R2.03.00.00 (Beta) (c) gedas 1999
        All rights reserved.
1999/07/22 12:57:52 I: <CMD_EXECUTED    > Kommando 'START /USER' erfolgreich aus
gefuehrt.
SEND /C DSN=c:\temp\liesmich.txt (SID=LOC DSNNEW=neu.txt)
1999/07/22 12:59:43 >>> SEND /C DSN=c:\temp\liesmich.txt (SID=LOC DSNNEW=neu.txt
)
1999/07/22 12:59:43 I: <CMD_EXECUTED    > Kommando 'SEND /CREATE' erfolgreich au
sgefuehrt.
1999/07/22 13:02:21 I: <CMD_EXECUTED    > Kommando 'END' erfolgreich ausgefuehrt
.

c:\rvssrc>^C
c:\rvssrc>_
```

## 10.2 Die C-CAL-Schnittstelle verwenden

### 10.2.1 Kompilieren und binden der C-CAL-Schnitttelle für rvsNT

C-CAL-Schnittstelle ist getestet und geschrieben in Microsoft Visual C++. Die Prototypdefinitionen befinden sich in der Header-Datei `rvscal.h`. Die folgende Zeile muss das Anwendungsprogramm enthalten:

- `#include rvscal.h`

Es wird noch eine Header-Datei `rixstd.h` benötigt, die aber nicht explizit inkludiert werden muss. Wir empfehlen Run-Time-Linking zum Binden der C-CAL-Schnittstelle zu verwenden. Daher muss Ihre Anwendung folgende Aufrufe absetzen:

- `LOAD_RVSCALL_DLL (HANDLE &hlib)`. Das ist ein Makro zum Laden der korrekten Version der dynamischen Bibliothek von rvsNT. Dieser Makro ruft indirekt die Funktion `LoadLibrary()` und lädt so die aktuelle Version der rvsNT-Bibliothek. Der Parameter `&hlib` ist die Adresse des handles der DLL. Wir empfehlen diesen Makro zu benutzen, um kompatibel zu späteren Versionen der rvs-Biblithek sein zu können.
- `GetProcAddress()` .Mittels `GetProcAddress()` holt man sich die Adresse einer rvsNT-Funktion, die man aufrufen möchte z.B. `rvsCreateSendEntry()`.
- `FreeLibrary()` sollten Sie aufrufen, nachdem Die die Benutzung von der C-CAL-Schnittstelle beendet haben.

#### Optionen zum Kompilieren und Binden

- **Packing**: Die structs der C-CAL-Schnittstelle werden an 4-Byte-Grenzen ausgerichtet. Daher muss die Anwendung mit der Option `/Zp4` kompiliert werden.
- **char type**: Die C-CAL-Schnittstelle benutzt `unsigned char` als default char type. Daher muss auch die Applikation mit der dies bewirkenden Kompilieroption `/J` kompiliert werden.

### Bibliothekdateien

Für Ihre Anwendung, die C-CAL-Schnittstelle benutzt, benötigen Sie zwei dynamischen Bibliotheken von rvsNT:

- RVSCALL.DLL , die von der Anwendung geladen wird.
- RVSCALL22.DLL, die zur Laufzeit von RVSCALL.DLL geladen wird; darf nicht von der Applikation geladen werden.

**Beispiel:**

```
#include <windows.h>
#include <stdio.h>
#include "rvscal.h"
void main(void)
{
    HANDLE  hRvsLib;
    char        str[128];
    FARPROC     prvsGetDBVersion;

    LOAD_RVSCAL_DLL(&hRvsLib);
    if (!hRvsLib)
    {
            printf("Error in LoadLibrary, rc= %d\n", GetLastError());
            return;
        }

    prvsGetDBVersion = GetProcAddress( hRvsLib, "rvsGetDBVersion");
    if (!prvsGetDBVersion)
    {
            printf("Error in GetProcAddress rvsGetDBVersion, rc= %d\n",
GetLastError() );
            return;
        }

    prvsGetDBVersion(str);
    printf("Version: %s\n", str);
}
```

### 10.2.2 C-CAL-Schnittstelle für UNIX und OS/400

Für den Einsatz der C-CAL-Schnittstelle müssen Sie folgende Bibliothek binden:

- rpulib.a für **UNIX**-Systeme (statisches Binden). Diese Bibliothek befindet sich im Verzeichnis $RVSPATH/system. Das Programm $RVSPATH/system/rvscd demonstriert die übliche Nutzung der C-CAL-Schnittstelle. Den C-Quell-Code für dieses Programm finden Sie in der Datei $RVSPATH/samples/s_rvscd.c und die dazugehörige make-Datei s_make.cd im Verzeichnis $RVSPATH/system.
- rvscal für **OS/400** (Service-Programm). Dieses Programm befindet sich in der Bibliothek RVS_SYSTEM.

Dann muss das benutzerdefinierte C-Programm, das diese Bibliothek benutzt, die Header-Datei rvscal.h in den Quell-Code einbinden.

**Prototyp der Funktion rvscal**

```
int rvscal(char s_cmd[],
```

```
int *p_l_cmd,
char *p_c_msglvl,
char s_msg[],
int *p_l_msg,
long *p_cmdid);
```

**Hinweis**: Die Parameterliste wurde so gestaltet, dass auch Aufrufe durch andere Programmiersprachen möglich sind.

## Beschreibung der Parameter

`RETURNvalue` (int)

=**0**, wenn `rvscal` erfolgreich war (obwohl der Nachrichtentext abgeschnitten sein kann); von **0** verschieden, wenn ein Fehler auftrat.

Es gibt zwei Arten von Fehlercodes:

- Fehler die vom Kommando-Parser wegen einer ungültigen Kommandosyntax ausgelöst wurden. Die Fehlercodes liegen im Bereich zwischen 1 und 7. Eine genaue Beschreibung finden Sie im "Meldungs- und Return-Code-Handbuch".
- Andere Fehler, die auftraten, während rvs® versuchte, das angegebene Kommando auszuführen. Eine Beschreibung der Fehlerursache wird mit `s_msg` zurückgegeben.

`s_cmd` (char [], input)

Zeiger auf das Character-Array mit dem Kommando-String; die Größe des Array muss mindestens *****`p_l_cmd+1` sein.

`p_l_cmd` (int *, input)

Länge des Kommando-Strings; `s_cmd` muss null-terminiert sein, wenn *****`p_l_cmd` größer als die Länge des Kommando-Strings ist.

`p_c_msglvl` (char *, output)

gibt das Level der Nachricht zurück; möglich sind **I** (informativ), **W** (Warnung), **E** (error, Fehler), **S** (severe, schwerer Fehler)

`s_msg` (char [], output)

Zeiger auf den Character-String der Mindestlänge *****`p_l_msg+1`; der Character-String erhält eine Nachricht von `rvscal`, die Informationen über den Erfolg des Aufrufs von `rvscal` gibt.

Dieser Abschnitt beschreibt die Syntax der Kommandos sowie den von der C-CAL-Schnittstelle (`rvscal`) und der Kommandozeilen-Schnittstelle (`rvsbat`) benutzten gültigen Kommandosatz.

Die unten stehenden Beispiele benutzen zwei Syntax- Erweiterungen, die nur für die Kommandozeilen-Schnittstelle verfügbar sind:

- Kommentarzeilen beginnen mit einem ***** in der Spalte **1**,

- Kommandos können in der nachfolgenden Zeile weitergeführt werden, indem ein **+** als letztes Zeichen der fortzusetzenden Zeile steht.

## 10.3 Syntax der Kommandos

Ein Utility-Kommando muss der untenstehenden Syntax folgen:

- Es besteht aus
  - einem Kommando-Verb,
  - einem optionalen Bestimmungswort wie `/CREATE, /DELETE`, usw. Das Bestimmungswort kann mit **/** oder **–** beginnen und darf auf einen Buchstaben abgekürzt sein.
  - Werte nicht-wiederholbarer Parameter, angegeben durch <Parametername>=<Parameterwert>. Vor und nach dem Gleichheitszeichen "=" darf kein Leerzeichen stehen.
  - Werte wiederholbarer Parameter (nur Kommando `send`); diese folgen denselben Syntaxregeln wie für nicht-wiederholbare Parameter. Eine Gruppe wiederholbarer Parameter wird in Klammern gesetzt. Es gibt beliebig viele Gruppen wiederholbarer Parameter.
- Kommando-Verb, Bestimmungsworte und Parameternamen können in Groß-, Kleinbuchstaben oder gemischt angegeben sein.
- Parameterwerte werden in Großbuchstaben konvertiert, wenn sie nicht durch einfache oder doppelte Anführungszeichen geschützt sind.
  Wenn ein Parameterwert das schützende Zeichen selbst enthalten soll, muss das schützende Zeichen zweimal angegeben sein. Z.B. sind die folgenden Angaben gleichbedeutend.

  `PARM=' "test string" ' or parm="""test string"""`
  Nicht geschützte Parameterwerte können beliebige alphanumerische Zeichen enthalten. Sie können einen Satz spezieller Zeichen nicht enthalten, z.B. Leerzeichen, einfache oder doppelte Anführungszeichen und Klammern.

- Unterschiedliche Parameterangaben müssen durch mindestens ein Leerzeichen voneinander getrennt sein.

## 10.4 Das Kommando `START`

### Funktion:

- Eine Sitzung mit den rvs® Utilities starten
- rvs® Datenbank öffnen,
- Prüfen, ob der aktuelle Benutzer die rvs® Utilities benutzen darf
- `start /USER` wird implizit durch die C-CAL-Schnittstelle aufgerufen, wenn es nicht explizit aufgerufen wird. Wenn die Kommandozeilen-Schnittstelle die Ausführung startet, wird `start /USER` immer implizit aufgerufen.

### Bestimmungsworte:

`/USER` (Standard) startet die Sitzung für den rvs® Benutzer

**Parameter:**

**RVSENV** (optional) Name der rvs® Umgebungsdatei

**Beispiele:**

Umgebungsdatei in (rvs400 Bibliothek) **RVS_NEW/DAT(RVSENV)** benutzen:

```
START /USER RVSENV="RVS_NEW/DAT(RVSENV)"
```

## 10.5 Das Kommando `END`

**Funktion:**

- Beendet die Sitzung mit den rvs® Utilities
- Schließt die rvs® Datenbank
- `END` muss bei der Verwendung der C-CAL-Schnittstelle aufgerufen werden. Wenn die Kommandozeilen-Schnittstelle die Ausführung startet, wird `END` immer implizit aufgerufen.

**Bestimmungsworte:**

keine

**Parameter:**

keine

**Beispiele:**

```
END
```

## 10.6 Das Kommando `SEND`

**Funktion:**

- Erstellt
- ändert oder
- löscht

einen Sendeauftrag für die Übertragung einer lokalen Datei an einen anderen rvs® Knoten.

**Bestimmungsworte:**

`/CREATE` (Standard) erstellt einen Sendeauftrag

| | |
|---|---|
| oder `/C` | |
| `/DELETE` oder `/D` | löscht einen Sendeauftrag |
| `/HOLD` oder `/H` | setzt den wartenden Sendeauftrag in den Status "angehalten" (held) |
| `/RELEASE` oder `/R` | Sendeauftrag freigeben, der zuvor in den Status "angehalten" gesetzt wurde |

**`SEND /CREATE` Parameter:**

| | |
|---|---|
| **DSN** | (erforderlich) Name der zu sendenden lokalen Datei, hier ist der ganze Pfad der Datei anzugeben. |
| **CODEIN** | (optional) Code der lokalen Datei (**A**=ASCII, **E**=EBCDIC);<br>Standard: lokaler Code des Systems. |
| **DISP** | (optional) Disposition für die lokale Datei, nachdem der Sendeauftrag erfolgreich ausgeführt ist: **K**=halten (keep, Datei wird nicht gelöscht nach dem Versand, Standard), **D**=löschen (delete, Datei wird gelöscht nach dem Versand). |
| **FORMAT** | (optional) das für die Übertragung mit ODETTE benutzte Format:<br>**T**=Text (eine Folge von ASCII-Zeichen), **U**= unstrukturiert (binär), **F**=feste Satzlänge, **V**=variable Satzlänge;<br>Standard: Satzformat der lokalen Datei<br>(**U** für rvsNT, rvsXP und rvsX, **F** für rvs400) |
| **ACCOUNT** | Berechnungsnummer oder Code. |
| **INITTIME** | (optional) frühest möglicher Zeitpunkt, zu dem der Sendeauftrag ausgeführt werden kann; möglich sind: **H**=anhalten (hold), **N**=jetzt (now, Standard), oder eine bestimmte Zeit im Format **JJJJ/MM/TT HH:MM** (**Beispiel**: 2004/09/04 10:43). |
| **SERIAL** | (optional) bei **Y** (=Ja) wird der Sendeauftrag in der von Ihnen eingegebenen Reihenfolge gesendet (siehe auch **LABEL**). Die Datei wird erst gesendet, wenn der vorhergehende Sendeauftrag vollständig erledigt ist. |
| **LABEL** | (optional) Benutzerkennung (bis zu 20 Zeichen) für die Serialisierung eines vorangehenden Sendeauftrages (bei **SERIAL**=**Y**). Alle Dateien, die in der gleichen Gruppe versendet werden, müssen die gleiche Kennung (**LABEL**) haben. |
| **VFTYP** | (optional) Textdateien können auch im Format Fest(**F**) oder Variable(**V**) versendet werden, ohne mit dem rvs®- |

Hilfswerkzeug `rvsut2fv` (siehe Abschnitt `9.1`) bearbeitet zu werden.

**VFTYP=T** (text) bedeutet, dass Ihre Textdatei ohne `rvsut2fv` versendet wird.

Textdatei bedeutet eine Folge von ASCII-Zeichen, wobei bei den einzelnen Satzlängen, der Zeilenwechsel (CR/LF bei MS Windows, LF bei UNIX-Systemen) nicht dazugerechnet wird. Neben dem Parameter **VFTYP** müssen auch die Parameter **MAXRECL** und **FORMAT** gesetzt werden.
**Beispiel**: Wenn Sie eine Textdatei im Format **F** mit Satzlänge 80 versenden möchten, muss Ihre Textdatei in jeder Zeile 80 Zeichen lang sein (ohne CR/LF oder LF). Folgende Parameter sind dann wie folgt zu setzen:
**VFTYP=T**, **FORMAT=F**, **MAXRECL=80**.
Wenn Sie aber Ihre Textdatei im Format **V** versenden möchten, bedeutet dies, dass Sie eine Textdatei (eine Folge von ASCII-Zeichen) mit unterschiedlichen Zeilenlängen vorbereitet haben (z.B. der längste Satz ist 120 Zeichen lang) und die notwendigen Parameter sind dann wie folgt zu belegen:
**VFTYP=T**, **FORMAT=V**, **MAXRECL=120**. Als **MAXRECL** ist die maximale Länge eines Satzes bis zum Zeilenwechsel gedacht (ohne CR/LF oder LF). Siehe auch Beispiele am Ende des Abschnittes.

**VFTYP=V** (intern) bedeutet, dass die Datei vor dem Versand mit `rvsut2fv` behandelt wurde. Dann ist nur das Format (**FORMAT**) der zu sendenden Datei anzugeben, ohne den Parameter **MAXRECL**. Siehe Beispiele am Ende des Abschnittes.

**VFTYP=X** bedeutet, dass eine Texdatei automatisch beim Versenden mit `rvsut2fv` konvertiert wird. Beispiel: **VFTYP=X**, **FORMAT=F**, **MAXRECL=80**. Der Unterschied zum VFTYP=T ist, dass die Datei nicht die gewünschte Zeilenlänge besitzen muss, dies wird vom `rvsut2fv` erledigt.

**VFTYP=U** bedeutet, dass eine binäre Datei automatisch beim Versenden mit `rvsut2fv` konvertiert wird. Die gewünschte Zeilenlänge muss nicht vorhanden sein; dies wird vom `rvsut2fv` erledigt. Beispiel: **VFTYP=U**, **FORMAT=F**, **MAXRECL=80**.

**VFTYP=D** bedeutet, dass Dateien im Format **Fest** als Binärdateien übertragen werden und Dateien im Format **Variabel** als Textdateien übertragen werden.

**MAXRECL** (optional) maximale Satzlänge für Dateien im Format **T**, **F** oder **V**; dieser Parameter wird nur benötigt, wenn die Dateien nicht mit dem rvs®-Hilfswerkzeug `rvsut2fv` davor konvertiert wurden. Siehe auch Parameter **VFTYP**.

**SIDORIG** ID der virtuellen Station beim Senden; der Wert dieses Parameters wird beim Senden in die SFID (Start File ID, siehe Kapitel 3.3) eingefügt.

Die folgende Gruppe der Sendeparameter sind innerhalb der runden Klammern () anzugeben:

**SID** (erforderlich) Stations-ID des Empfängers

**UID** (optional) Benutzer-ID des Empfängers; wenn kein Wert oder einer leerer String angegeben ist, ist dies des System auf dem fernen Station

**COMPRESSION** (optional) bei **Y** (=Ja) wird die Datei vor dem Senden komprimiert. Dieser Parameter muss beim Aufruf nach dem **SID** Parameter in Klammern angegeben werden; z.B.

```
SEND /C DSN=/home/test/test11.txt (SID=RTZ
COMPRESSION=Y)
```

**ENCRYPTION** (optional) bei **Y** (=Ja) wird die Datei vor dem Senden verschlüsselt. Dieser Parameter muss beim Aufruf nach dem **SID** Parameter in Klammern angegeben werden; z.B.

```
SEND /C DSN=/home/test/test11.txt (SID=RTZ
COMPRESSION=Y ENCRYPTION=Y)
```

**DSNNEW** (optional) Für die Übertragung benutzter virtueller Dateiname (VDSN).

Bei Übertragungen an einen MVS-Hostrechner, muss dies ein gültiger MVS-Dateiname sein (den RACF-Regeln entsprechen).

Standard: VDSN wird aus dem lokalen Dateinamen gebildet. Die maximale Länge von VDSN ist nach ODETTE 26 Zeichen.

**CODEOUT** (optional) Gewünschter Code (**A**=ASCII, **E**=EBCDIC) der Datei beim Empfänger. Z.B.

```
send /c dsn=C:\test22.dat codein=a
format=v(sid=rtt codeout=e
dsnnew=FIX0GBE.TEXT)
```

**CODETABLE** (optional) Definiert die Codetabelle, die zur

Codeumwandlung (siehe auch Abschnitt über Codeumwandlung im Benutzerhanbuch) verwendet werden soll. Hier ist der ganze Pfad der Codetabelle-Datei anzugeben.
**Beispiel**: `send /c dsn=/home/send/test22.dat codein=a format=v (sid=rtt codeout=e codetable=/home/tables /rtcusrdat dsnnew=FIX0GBE.TEXT)`

**`SEND /DELETE, /HOLD, /RELEASE` Parameter:**

**CMDID** (optional) Eindeutige Kommandonummer des Sendeauftrages.

Der zu bearbeitende Sendeauftrag wird durch die eindeutige Kommandonummer **CMDID** des Sendeauftrags identifiziert.

Die folgenden Beispiele zeigen, wie Kommandos in einer Datei angegeben werden können, die als Eingabe-Datei für `rvsbat` benutzt wird. Der Gebrauch von Fortsetzungszeilen (die vorhergehende Zeile endet mit **+**) und Kommentaren (* in Spalte 1) wird ebenso gezeigt.

**Beispiele:**

```
SEND /C DSN=C:/RVS/LPDBI.C +
SERIAL=n LABEL=l1 inittime=NOW +
CODEIN=A FORMAT=T DISP=d +
(SID=st1 UID=user1 CODEOUT=e DSNNEW=dsnnew1)

SEND /C DSN=/HOMe/LPDBI.C +
SERIAL=y LABEL=l1 inittime=HOLD +
FORMAT=U DISP=k (SID=st1)

SEND /C DSN=rpu.c +
SERIAL=y LABEL=l1 inittime='1991/07/01 10:35' +
FORMAT=U (SID=st1)

*------ serialize on data set (without specifying * the full
data set name)

SEND /C DSN=lpdbi.c SERIAL=y FORMAT=T (SID=st1)

*------ serialize again on data set (and default * for FORMAT)
*
SEND /C DSN=lpdbi.c SERIAL=y (SID=st1)
```

```
* --- Textdatei zum Host senden ohne rvsut2fv
send /c dsn=C:\test.text codein=a format=f vftyp=t
maxrecl=80(sid=rtt codeout=e dsnnew=FIX0GBE.TEXT)

* --- Textdatei, die mit rvsut2fv konvertiert
* --- wurde zum Host senden
send /c dsn=C:\test22.text codein=a format=v
(sid=rtt codeout=e dsnnew=FIX0GBE.TEXT)
```

## 10.7 Das Kommando `RESENTR`

**Funktion:**

Erstellen, Überarbeiten und Löschen von residenten Empfangseinträgen.

**Bestimmungsworte:**

| | |
|---|---|
| `/CREATE` oder `/C` | (Standard) erstellt einen residenten Empfangseintrag |
| `/UPDATE` oder `/U` | überarbeitet einen residenten Empfangseintrag |
| `/DELETE` oder `/D` | löscht einen residenten Empfangseintrag |

**Parameter:**

| | |
|---|---|
| **DSN** | (erforderlich) Virtueller Name der eintreffenden Datei (VDSN, darf max. 26 Zeichen lang sein). |
| **SID** | (erforderlich) Stations-ID des Senders. |
| **ACCOUNT** | (optional) Berechnungsnummer oder Code |
| **COMMENT** | (optional) Kommentar zur Beschreibung des residenten Empfangseintrages (bis zu 50 Zeichen lang);<br>Standard: leerer String |
| **DSNNEW** | (optional) der neue Dateiname auf dem lokalen System. Hier ist der ganze Pfad der neuen Datei anzugeben.<br>Standard: Der lokale Dateiname wird aus dem virtuellen Dateinamen gebildet. |
| **JOB** | Name einer Kommandozeilen-Datei, die nach der Zustellung der Datei gestartet wird[5] (`.bat` unter Windows-Systemen; `.sh` für UNIX-Systeme).<br>Diese Kommandozeilen-Datei darf Ersetzungsvariablen |

[5] Die Art des geforderten Jobs hängt vom Betriebssystem ab. Z.B. wird unter OS/2 eine CMD-Datei als Hintergrundprozeß ausgeführt, während unter OS/400 rvs® ein Job mit Hilfe von **SBMDBJOB** startet.

enthalten. rvs® ersetzt sie, bevor es den Job zur Ausführung an das Betriebssystem übergibt. Folgende Ersetzungsvariablen sind möglich:

- ?**DSN**?: Name der lokalen Datei.
- ?**VDSN**?: Virtueller Dateiname, unter welchem die Datei übertragen wurde.
- ?**DTAVAIL**?: Datum, zu dem die Datei für das Senden verfügbar war.
- ?**FORMAT**?: Satzformat der empfangenen Datei.
  - **F** feste Satzlänge
  - **V** variable Satzlänge
  - **T** Textdatei (ASCII-Zeichenfolge)
  - **U** unstrukturierte Datei (binär)
- ?**MAXRECL**?: Die Bedeutung dieses Feldes hängt vom Satzformat der empfangenen Datei ab.
  - **F** Format: Länge jedes Satzes.
  - **V** Format: Maximale Länge, den ein Satz haben kann.
  - **T** und **U** Format: Immer **0** (null)
- ?**BYTES**?: Zahl der übertragenen Bytes
- ?**RECORDS**?: Zahl der übertragenen Sätze für Dateien im **F** und **V** Format; für Dateien im **T** und **U** Format immer **0** (null).
- ?**DTRCV**?: Datum, an dem die Datei dem lokalen Benutzer zugestellt wurde.
- ?**LUID**?: ID des (lokalen) Empfängers
- ?**UID**?: ID des Senders
- ?**SID**?: Stations-ID des Senders
- **?SIDDEST?** StationsID der virtuellen Empfangsstation
- ?**CNQS**? Kommandonummer der QuittungsSendung (EERP) für die empfangene Datei.
- ?**CNIE**? Kommandonummer des InformationsEingangs (IE) für die empfangene Datei
- ?**CNIZ**? Kommandonummer der InformationsZustellung (IZ) für die empfangene Datei
- ?**DSNTEMP**?: Name der temporären Datei (Diese Datei kann am Ende des Sendeauftrages mit dem Befehl

– `DELETE ?DSNTEMP?` (**NT**, **XP**, **2000**)

– `rm ?DSNTEMP?` **(UNIX)**

– `DLTF ?DSNTEMP?` (**OS/400**)

gelöscht werden.)

**DISP** (optional) Bestimmt die Behandlung der Datei, wenn die Bearbeitung abgeschlossen ist.

- **K** hält die Datei
- **D** löscht die Datei: Wichtiger Hinweis: Wenn Sie sich für die Option D entscheiden, wird die Datei auf jeden Fall gelöscht und nur als Trigger/Schalter für einen residenten Empfangseintrag verwendet. Es sind keine weiteren Operationen mit der empfangenen Datei möglich.

Standard: **K**

**REPLACE** (optional) Wenn **DISP=K** (halten) eingestellt ist, legt **REPLACE** die Aktionen fest, die rvs® durchführt, nach dem Empfang einer Datei, die im Namen mit einer lokalen Datei übereinstimmt.

**R** Vorhandene Datei ersetzten

**N** Neuen eindeutigen Dateinamen verwenden

**I** Eintreffen der Datei ignorieren

Standard: **N**

**TSTAMP** (optional) Kann **Y**=Ja oder **N**=Nein sein; teilt rvs® mit, ob der Dateiname zur eindeutigen Kennzeichnung mit Zeitstempel versehen werden soll, wenn die Datei eingetroffen ist.

Standard: **N**

**CODETRANS** (optional) Gibt an, ob die empfangene Datei mit rvs®-internen Codeumwandlungstabellen in ASCII oder EBCDIC konvertiert werden soll, oder ob zu Codeumwandlung eine eigene Codetabelle benutzt werden soll (siehe auch Kapitel über Codeumwandlung im rvs® Benutzerhandbuch).

**E** für die Codeumwandlung von EBCDIC nach ASCII

**A** für die Codeumwandlung von ASCII nach EBCDIC

**T** für die Codeumwandlung mit Codetabelle

z.B.
```
resentr /c dsn="<empfangene ASCII Datei>"
codetrans=a sid="<Sender>"
```

**CODETABLE** (optional) Definiert die eigene Codeumwandlungstabelle (siehe auch Abschnitt über Codeumwandlung im Benutzerhanbuch), die verwendet werden soll.

z.B.
```
resentr /c dsn="<empfangene EBCDIC Datei>"
codetrans=t codetable ="<User Code Tabelle,
z.B. /home/rvs/arcdir/rtcusrdat>"
sid="<Sender>"
```

**VFTYP** (optional) Sie können hier bestimmen, ob die empfangene Datei als Textdatei mit Zeilenwechsel (CR/LF bei MS Windows, LF bei Unix-Systemen) nach jedem Satz abgelegt werden soll. Diese Angabe gilt nur für Dateien, die im Format **Fixed** oder **Variable** empfangen werden.

**VFTYP=T** (text) bedeutet, dass die empfangene Datei als Textdatei mit Zeilenwechsel gespeichert werden soll.

**VFTYP=V** (variabel) bedeutet, dass die empfangene Datei im Format **Fixed** oder **Variable** ohne Umwandlung in Textdateien im rvs® internen Format gespeichert wird.

**VFTYP=S** (Sinix) bedeutet, dass die empfangene Datei im Sinix-Format gespeichert wird.

**Beispiele:**

```
*
*---------- use all parameters
*
RESENTR /C DSN=incoming1  SID=st2 +
 DSNNEW=/home/local.dsn REPLACE=n DISP=k +
 JOB=/home/rvs/bin/rcv.sh COMMENT='This is a test RE'
*
*---------- use defaults
*
RESENTR /C DSN=incoming2  SID=st2 +
 REPLACE=i    DISP=d
*
*---------- no UID, DISP
*
RESENTR /C DSN=incoming3  SID=st2 +
 REPLACE=r
*
*---------- delete RESENTR
*
RESENTR /D DSN=incoming3  SID=st2
*
*---------- update RESENTR
*
RESENTR /U DSN=incoming2  SID=st2 +
REPLACE=n    JOB=/home/rvs/bin/rcv.sh
```

## 10.8 Kommando `SENDJOB`

### Funktion:

Erstellen, Überarbeiten und Löschen von Einträgen für Jobstart nach Sendeversuch.

### Bestimmungsworte:

| | |
|---|---|
| `/CREATE` | (Standard) erstellt einen Eintrag für Jobstart nach Sendeversuch |
| `/UPDATE` | überarbeitet einen residenter Empfangseintrag |
| `/DELETE` | löscht einen residenten Empfangseintrag |

### Parameter:

**VDSN** (erforderlich) Virtueller Name der gesendeten Datei.

**SID** (erforderlich) Stations-ID des Empfängers

**ATTEMPTS** (optional) Zahl der Sendeversuche; legt fest, bei welcher Bedingung der Job starten soll

Wert **0**: Der Job startet bei erfolgreicher Übertragung

Wert >**0**: Der Job startet, wenn die angegebene Zahl fehlgeschlagener Sendeversuche erreicht ist

Standard: **0**

**JOB** Name des Jobs, der nach erfolgreicher Übertragung oder erfolglosen Sendeversuch gestartet werden soll. Diese Kommandozeilen-Datei (der Job) darf Ersetzungsvariablen enthalten. rvs® ersetzt sie, bevor es den Job zur Ausführung an das Betriebssystem übergibt:

- ?**DSN**?: Name der lokalen Datei, die gesendet wurde. Bei **EERP** gibt es keinen lokalen Dateinamen. Der Wert von ?**DSN**? hat die Form **QS** ( **SIDORIG** - **SIDDEST**) mit der Bedeutung: **SIDORIG** Stations-ID des Senders; **SIDDEST** Stations-ID des Empfängers.
- ?**VDSN**?: virtueller Dateiname unter dem die Datei übertragen wurde.
- ?**DTAVAIL**?: Datum, an dem die Datei für das Senden verfügbar war.
- ?**FORMAT**?: Satzformat der gesendeten Datei
  - **F** fest
  - **V** variabel
  - **T** Text
  - **U** unstrukturiert

- ?**BYTES**?: Zahl der übertragenen Bytes
- ?**RECORDS**?: Zahl der im Format **F** oder **V** übertragenen Sätze
- ?**DTRCV**?: Datum, an dem die Datei dem lokalen Benutzer zugestellt wurde.
- ?**LABEL**?: Zeichenfolge, wenn das Sendekommando ein **LABEL** Parameter enthielt.. Kann zur Identifikation des Sendekommandos benutzt werden.
- ?**SECN**?: Kommandonummer des Sendekommandos **SE**. Kann zur Identifikation des Sendekommandos benutzt werden.
- ?**SKCN**?: Nummer des Sendekommandos **SK**.
- ?**UID**?: Benutzer-ID des Senders; bei **EERP** ist der Wert immer "!-**QS**-!”.
- ?**SID**?: Stations-ID des Empfängers
- ?**SIDORIG**? Stations-ID der virtuellen Senderstation.
- ?**SENDATT**? Anzahl der Sendeversuche, nach denen der Job gestartet werden soll.
- ?**DSNTEMP**?: Name der temporären Datei (Diese Datei kann am Ende des Sendeauftrages mit dem Befehl

– `DELETE ?DSNTEMP?` (**NT**)

– `rm ?DSNTEMP?` **(UNIX)**

– `DLTF ?DSNTEMP?` (**OS/400**) gelöscht werden.)

**COMMENT** (optional) Ein Kurztext zur Beschreibung des Zwecks des Jobstarteintrages (bis zu 50 Zeichen lang);

Standard: leerer String

**Beispiele:**

```
*
*---------- use all parameters
*
SENDJOB /C VDSN=sending1  SID=st2 ATTEMPTS=1 +
 JOB=/home/rvs/bin/send-fail1.sh COMMENT='This is a test JS'
*
*---------- use defaults
*
SENDJOB /C VDSN=sending2  SID=st2 JOB=/home/rvs/bin/snd.sh
*
*---------- Job should start after data have been transmitted
successfully
*
SENDJOB /C VDSN=sending3  SID=st2 JOB=/home/rvs/bin/snd.sh
*
```

```
*---------- delete SENDJOB
*
SENDJOB /D VDSN=sending3  SID=st2
*
*---------- update SENDJOB
*
SENDJOB /U DSN=/home/sending2  SID=st2
JOB=/home/rvs/bin/sendok.sh
```

## 10.9 Das Kommando `USER`

**Funktion:**

- rvs® Benutzertabelle ändern
- Benutzer einstellen
- Dialogsprache für Benutzer festlegen
- Privilegien für die Benutzer festlegen

**Bestimmungsworte:**

| | |
|---|---|
| `/CREATE` | erstellt einen Eintrag in die rvs® Benutzertabelle |
| `/DELETE` | löscht einen Eintrag in die rvs® Benutzertabelle |
| `/UPDATE` | überarbeitet einen Eintrag in die rvs® Benutzertabelle |

Wenn kein Bestimmungswort angegeben ist, wird der Eintrag erstellt oder überarbeitet, je nachdem, ob der Eintrag schon besteht oder nicht.

**Parameter:**

| | |
|---|---|
| **UID** | (optional) Benutzer-ID, die den zu ändernden Eintrag identifiziert; Sie können nur als privilegierter Benutzer etwas anderes als die aktuelle Benutzer-ID angeben.<br>Standard: current user |
| **NAME** | (optional) neuer Benutzername |
| **LANGUAGE** | (optional) Neue Sprache angeben (z.B. **E**=Englisch, **D**=Deutsch) |
| **PRIV** | (optional) neues Benutzerprivileg; (**U**=Benutzer, **O**=Operator, **A**=Administrator) |

**Beispiele:**

```
*---------- create new user (default=own UID)
*
USER /C
*
*---------- create new user
*
USER /C UID=newuser LANGUAGE=d NAME=x PRIV=O
USER /C UID=extrauser
*
```

```
*---------- update existing user
*
USER /U UID=newuser LANGUAGE=e NAME=y PRIV=U
*
*---------- delete user
*
USER /D UID=newuser
```

## 10.10 Das Kommando ACTIVATE

**Funktion:**

- Aktiviert eine Partnerstation
- rvs® startet einen Sendeprozess, der die Verbindung zur Partnerstation herstellt. Alle Dateien in Wartestellung werden übertragen. Danach wird die Verbindung wieder getrennt.

**Parameter:**

**SID** (erforderlich) Stations-ID des Partners

**Beispiele:**

```
*
*---------- activate station ABC
*     (connect and send or receive queued data sets)
*
ACTIVATE SID=ABC
```

## 10.11 Das Kommando `MODST`

**Funktion:**

Stationstabelle ändern.

**Achtung: Dieses Kommando überschreibt alte Datenbankeinträge**; andere vorhandene Einträge werden nicht gelöscht () nur **DELST** löscht einen Eintrag).

**Parameter:**

**DSN** (erforderlich) Name der Datei, die die Stationstabelle enthält. Der Dateiname kann eine einfache Eingabedatei oder ein Verzeichnis mit mehreren darin enthaltenen Eingabedateien sein.

**Beispiele:**

```
*
*---------- modify station table:
*     (open the file (here: UNIX file name),
*     read contents and put it into database):
*
MODST DSN="/home/rvs/init/new_rdstat.dat"
```

## 10.12 Das Kommando `DELST`

**Funktion:**

Stationstabelle löschen.

**Parameter:**

**SID** Stations-ID der zu löschenden Station

**Beispiele:**

```
*
*---------- delete entry ABC in database:
*
DELST SID=ABC
```

## 10.13 Das Kommando `LISTPARM`

**Funktion:**

- Gibt den Wert eines rvs® Parameters aus.
- Im Kommandozeilen-Modus erscheint die Meldung "I: Wert".

- In C-Programmen wird der Wert als String im `rvscal()`-Ausgabeparameter **S_MSG** übergeben.

**Parameter:**

**parameter** Name des rvs® Parameters.

**Beispiele:**

```
*
*---------- show the value of the parameter SLEEP
*
LISTPARM SLEEP
```

### 10.14 Das Kommando `SETPARM`

**Funktion:**

Setzt den Wert eines rvs® Parameters.

**Parameter:**

**parameter** Name des rvs® Parameters.

**Beispiele:**

```
*
*---------- set the value of the parameter SLEEP
*
SETPARM SLEEP=2
```

# 11 Arbeiten mit C-Funktionen

Die folgenden Abschnitte beschreiben, wie Sie die C-Sprachen-Funktionen benutzen, die Sie in ein Anwendungsprogramm einbinden können, um die rvs® Kommandos auszuführen. Bitte überprüfen Sie bei rvs® Updates `rvscal.h` nach letzten Änderungen von Strukturen und Funktionsprototypen.

## 11.1 Senden und Empfangen mit der C-CAL-Schnittstelle

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Sendeeinträgen mit der C-CAL-Schnittstelle erforderlich sind. Für alle Funktionen gelten folgende Typ-Definitionen und die zugehörigen Prototypen.

### 11.1.1 Typ-Definitionen

```
typedef struct {
 SINT i_error;
 SINT i_type;
 char s_uid [RVSCAL_L_USID];
 char s_jobid [RVSCAL_L_JOBID];
 char sid_neighb [RVSCAL_L_STATID];
 char sid_dest [RVSCAL_L_STATID]; /* destination SID */
 char sid_sender [RVSCAL_L_STATID];
 char s_vdsn [RVSCAL_L_VDSN];
 char dt_created [RVSCAL_L_DT]; /* job creation date and time
*/
 char dt_avail [RVSCAL_L_DT];
 char dt_sched [RVSCAL_L_DT];
 char dt_begin [RVSCAL_L_DT];
 char dt_end [RVSCAL_L_DT];
 char dt_done [RVSCAL_L_DT];
 char dt_received [RVSCAL_L_DT];
 long int cnt_sendatt; /* number of send attempts */
 long int cnt_record;
 long int cnt_byte; /*number of already sent bytes*/
 long int cnt_maxrecl;
 long int cnt_apsize;
 long int cnt_lenvm;
 long int cn_se;
 long int cn_sk;
 long int cn_ie;
 long int cn_iz;
 long int cn_re;
 char status_et [RVSCAL_L_KS];
 char status_se [RVSCAL_L_KS];
 char status_sk [RVSCAL_L_KS] ;/* state of send cmd (-
|a|i|f|p|e|d|s) */
 char status_ie [RVSCAL_L_KS];
 char status_iz [RVSCAL_L_KS];
 char dsn_local [RVSCAL_L_DSN];
 char s_recfm [RVSCAL_L_C1];
```

```
 char s_ftype [RVSCAL_L_C1];
 char s_code [RVSCAL_L_C1];
 char s_codein [RVSCAL_L_C1];
 char s_codeout [RVSCAL_L_C1];
 char s_disp [RVSCAL_L_C1];
 char s_label [RVSCAL_L_SEUSRLABEL]; /* user defined label */
 SINT flg_tstamp;
} INFO_SK ;


/*SetSendEntry Commands: */
#define SET_HOLD    1
#define SET_RELEASE  2
#define SET_DELETE   3


#define TRANSMISSION_SEND 1
#define TRANSMISSION_RECV 2


#define CN_START     0
```

### 11.1.2 Nächsten Sendeauftrag aus der Datenbank holen

**Prototyp `rvsGetNextSend`:**

```
PROCDEF int PROCKEYW rvsGetNextSend(long prev_send_cn, INFO_SK
*info);
```

**Beschreibung der Parameter**

| | |
|---|---|
| **FUNCTIONVALUE** | (int)<br>=**RVSCAL_OK**, wenn rvs® einen nächsten Sendeeintrag gefunden hat.<br>=**RVSCAL_END_FETCH**, wenn keine Sendeeinträge größer als **PREV_SEND_CN** vorhanden sind.<br>=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist |
| **prev_send_cn** | (int, input)<br>Kommandonummer des vorherigen Sendeeintrages |
| **info** | (struct **INFO_SK ***, output)<br>Struktur mit Informationen über den nächsten Eintrag mit einer Kommandonummer größer als **prev_send_cn** |
| **REMARKS** | `rvsGetNextSend` sucht nach dem nächsten Eintrag mit einem Wert größer als **prev_send_cn.** Wenn die Funktion das erste Mal aufgerufen wird, muß **prev_send_cn** |

**CN_START** sein**. CN_START** löste `rvsGetNextSend` aus, um Einträge von der rvs® Datenbank zu lesen und sie in einer internen Liste zu speichern. Jeder Aufruf von `rvsGetNextSend` mit **prev_send_cn**=**CN_START** aktualisiert diese Liste.

### 11.1.3 Einen Sendeauftrag aus der Datenbank holen

**Prototyp `rvsGetSendEntry`:**

```
PROCDEF int PROCKEYW rvsGetSendEntry(const long send_cn,
INFO_SK *info);
```

**Beschreibung der Parameter**

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn wir die Informationen über das Sendekommando gefunden haben

=**RVSCAL_END_FETCH**, wenn es keinen Sendeeintrag mit der angegebenen Nummer gibt

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler auftrat

**send_cn** (int, input)

Kommandonummer eines Sendeeintrages

**info** (struct **INFO_SK ***, output)

Struktur mit Informationen über den Sendeeintrag

### 11.1.4 Debug-Modus einschalten

**Prototyp `rvsSetDebugMode`:**

```
PROCDEF int PROCKEYW rvsSetDebugMode(int mode);
```

**Beschreibung der Parameter**

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn der Debug-Modus eingeschaltet ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

**MODE** (int, input)

Modustyp

### 11.1.5 Status von `SE` ändern

**Prototyp `rvsSetSendEntry`:**

```
PROCDEF int PROCKEYW rvsSetSendEntry(long int cn_se, int
SetCmd, char *szSID, char *s_msg);
```

**Beschreibung der Parameter**

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn kein Fehler aufgetreten ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler auftrat

**cn_se** (long int, input)

Kommandonummer des `SE` (zurückgegeben von `CreateSendEntry`)

**SetCmd** (int, input)

Kommandotyp

**SET_HOLD** : Diesen Sendeauftrag anhalten

**SET_RELEASE** : Diesen Sendeauftrag freigeben

**SET_DELETE** : Diesen Sendeauftrag löschen

**szSID** (char *, input)

Stations-ID des Empfängers

**s_msg** (char *, output)

Fehlermeldung bei einem Fehler

### 11.1.6 Nächsten Informationseintrag holen

**Prototyp `rvsGetNextIE`:**

```
PROCDEF int PROCKEYW rvsGetNextIE(long int prev_ie_cn, INFO_SK
*p_info);
```

**Beschreibung der Parameter**

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn kein Fehler aufgetreten ist.

| | |
|---|---|
| | =**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist |
| **prev_ie_cn** | (long int, input) |
| | Kommandonummer des vorhergehenden |
| **p_info** | (Struktur **INFO_SK** *, output) |
| | Datenstruktur mit Informationen über den Eintrag |

### 11.1.7 Eine Datei senden

**Prototyp `rvsCreateSendEntry`:**

```
PROCDEF int PROCKEYW rvsCreateSendEntry(
  char *dsn,
  char *disp,
  char *format,
  char *codein,
  char *inittime,
  char *serial,
  char *label,
  char *tstamp,
  char *sid,
  char *dsnnew,
char *codeout,
  char *s_msg);
```

**Beschreibung der Parameter**

| | |
|---|---|
| **FUNCTIONVALUE** | (int) |
| | =**RVSCAL_OK**, wenn die Funktion erfolgreich war |
| | =**RVSCAL_INTERNAL_ERROR**, wenn ein Fehler aufgetreten ist |
| **dsn** | (char *, input) |
| | Dateiname der lokalen Datei (Details siehe rvs® Benutzerhandbuch) |
| **disp** | (char *, input) |
| | Disposition (**K**=halten, **D**=nach Übertragung löschen) |
| **format** | (char *, input) |
| | Dateiformat (**T**/**F**/**V**/**U**) |
| **codein** | (char *, input) |
| | Code der Eingabedatei (**A**=ASCII, **E**=EBCDIC) |
| **inittime** | (char *, input) |

| | |
|---|---|
| | Zeitpunkt des frühest möglichen Sendeversuchs (**YY**/**MM**/**DD** **HH**:**MM**:**SS**) |
| **serial** | (char *, input) |
| | Serialisations-Flag (**Y** oder **N**) |
| **label** | (char *, input) |
| | Dateikennung (für Serialisation) |
| **tstamp** | (char *, input) |
| | Zeitstempel |
| **sid** | (char *, input) |
| | Stations-ID des Empfängers |
| **dsnnew** | (char *, input) |
| | Virtueller Dateiname (**OFTP**) |
| **codeout** | (char *, input) |
| | Ausgangs-Code (**A**=ASCII, **E**=EBCDIC) |
| **s_msg** | (char *, output) |
| | Fehlermeldung bei einem Fehler |

### 11.1.8 Sendeeintrag erstellen

**Prototyp `rvsCreateSendEntryCmd`:**

```
PROCDEF int PROCKEYW rvsCreateSendEntryCmd(
  char *dsn,
char *disp,
  char *format,
  char *codein,
  char *inittime,
  char *serial,
  char *label,
  char *tstamp,
  char *sid,
  char *dsnnew,
  char *codeout,
  char *s_msg);
```

**Beschreibung der Parameter**

| | |
|---|---|
| **FUNCTIONVALUE** | (int) |
| | =Kommandonummer des Sendeeintrages, wenn der Eintrag erfolgreich war |
| | =**RVSCAL_ERROR_CREATESEND**, wenn ein Fehler aufgetreten ist |
| **PARAMETERS** | siehe Kapitel "Eine Datei senden" |

## 11.2 Administration mit der C-CAL-Schnittstelle

Dieses Kapitel beschreibt die Funktionen zur Verwaltung von

- Stationseinträgen
- rvs® Parametern
- rvs® Operator-Kommandos
- residenten Empfangseinträge
- Einträgen für Jobstart nach Sendeversuch
- Benutzereinträgen
- Datenbank Funktionen

Für diese Funktionen steht eine Typ-Definition sowie die zugehörigen Prototypen und die Rückgabewerte zur Verfügung.

### 11.2.1 Funktionen zur Verwaltung von Stationstabellen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Stationstabelleneinträgen erforderlich sind.

#### 11.2.1.1 Typ-Definitionen

```
typedef struct {
 char      netid[RVSCAL_L_NETID];
 char      statname[RVSCAL_L_STATNAME];
 char      phone[RVSCAL_L_PHONE];
} INFO_ST;


typedef struct {
 char      ftp[RVSCAL_L_C1];
 char      protocol[RVSCAL_L_C1];
 char      autodial[RVSCAL_L_C1];
 SINT      pr_nk;
 SINT      flg_suspnd;
} INFO_NK;


typedef struct {
 char      sidneighb[RVSCAL_L_STATID];
 SINT      pr_rt;
} INFO_RT;


typedef struct {
 char      odetteid[RVSCAL_L_ODETTEID];
 char      pswfrom[RVSCAL_L_OPSW];
 char      pswto[RVSCAL_L_OPSW];
 long      i_sendblocks;
 long      i_recvblocks;
 long      i_ocreval;
```

```
 long      i_oexbuf;
 char      codein[RVSCAL_L_C1];
 char      codeout[RVSCAL_L_C1];
 char      userfield[RVSCAL_L_OFTP_USERD];
 char      eerp_in[RVSCAL_L_OFTP_EERP];
 char      eerp_out[RVSCAL_L_OFTP_EERP];
 char      vdsnchar[RVSCAL_L_OFTP_EERP];
 char      retry[RVSCAL_L_DT];
} INFO_OP;


typedef struct {
 char      netid_lu[RVSCAL_L_LU62_NETID];
 char      luname[RVSCAL_L_LU62_LUNAME];
 char      tpname[RVSCAL_L_LU62_TPNAME];
 char      userid[RVSCAL_L_LU62_UID];
 char      password[RVSCAL_L_LU62_PASSW];
 char      profile[RVSCAL_L_LU62_PROF];
 char      mode[RVSCAL_L_LU62_MODE];
 SINT      i_security;
 SINT      flg_sync;
 SINT      flg_conv;
} INFO_LU;


typedef struct {
 char      alias[RVSCAL_L_X25_ALIAS];
 char      recv_alias[RVSCAL_L_X25_ALIAS];
 long      cntn;
 char      xaddr[RVSCAL_L_X25_ADDR];
 char      subaddr[RVSCAL_L_X25_SUBADDR];
 char      timeout[RVSCAL_L_X25_TIMEOUT];
 char      isdnno[RVSCAL_L_X25_ISDNNO];
 char      link[RVSCAL_L_X25_LINK];
 char      fac[RVSCAL_L_X25_FAC];
 SINT      flgdbit;
 char      cug[RVSCAL_L_X25_CUG];
 SINT      flgreqrev;
 SINT      flgaccrev;
 SINT      flgfastsel;
 char      usrdata[RVSCAL_L_X25_USDTA];
 char      vc[RVSCAL_L_C1];
 long      cntsessions;
} INFO_XP;


typedef struct {
 char      protocol[RVSCAL_L_C1];
 long      cntn;
 char      inaddr[RVSCAL_L_TCPIP_ADDR];
 SINT      i_port;
 SINT      i_max_in;
 SINT      i_max_out;
 char      security[RVSCAL_L_TCPIP_SEC];
} INFO_TC;
```

```
typedef struct {
 char      lx_name[RVSCAL_L_LX_NAME];
 long      lx_len;
 char      lx_val[RVSCAL_L_LX_VALUE];
} INFO_LX;


typedef struct {
 char      sid[RVSCAL_L_STATID];
 int      flg_st;
 int      flg_nk;
 int      flg_rt;
 int      flg_op;
 int      flg_lu;
 int      flg_xp;
 int      flg_tc;
 int      flg_lx;
 INFO_ST    st;
 INFO_NK    nk;
 INFO_RT    rt;
 INFO_OP    op;
 INFO_LU    lu;
 INFO_XP    xp;
 INFO_TC    tc;
 INFO_LX    lx;
} INFO_STATION;
```

### 11.2.1.2 Nächsten Stationseintrag aus der Datenbank holen

**Prototyp `rvsGetNextStation`:**

```
PROCDEF int PROCKEYW rvsGetNextStation(char *SIDpre, char *
SID);
```

**Beschreibung der Parameter**

**SIDPRE** (char *, input)

**SID** der vorherigen Station

**SID** (char *, output)

**SID** der nächsten in **ST** (Stationstabelle) gefundenen Station

### 11.2.1.3 Stationseintrag in der Datenbank aktualisieren

**Prototyp `rvsUpdateStation`:**

```
PROCDEF int PROCKEYW rvsUpdateStation( INFO_STATION *info);
```

**Beschreibung der Parameter**

**info** (**INFO_STATION** *, input)

Struktur mit Informationen über den Stationseintrag

#### 11.2.1.4 Stationseintrag aus der Datenbank holen

**Prototyp `rvsGetStation`:**

```
PROCDEF int PROCKEYW rvsGetStation( char *psz_SID,
INFO_STATION *info);
```

**Beschreibung der Parameter**

**psz_SID** ( char *, input)

**SID** der Station

**info** (**INFO_STATION** *, output)

Struktur mit Informationen über den Stationseintrag

#### 11.2.1.5 Stationseintrag aus der Datenbank löschen

**Prototyp `rvsDeleteStation`:**

```
PROCDEF int PROCKEYW rvsDeleteStation( char *psz_SID);
```

**Beschreibung der Parameter**

**psz_SID** (char *, input)

SID des Stationseintrages

#### 11.2.1.6 Alle unterbrochenen Kommandos wieder aufnehmen

**Prototyp `rvsFreeSuspendedCommands`:**

```
PROCDEF int PROCKEYW rvsFreeSuspendedCommands( void);
```

**Beschreibung der Parameter**

keine Parameter vorhanden

#### 11.2.1.7 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist.

=**RVSCAL_END_FETCH**, wenn es keine Station mit dieser **SID** oder keinen Eintrag größer als **SIDPRE** gibt

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet werden konnte

=**RVSCAL_NEITHER_X_NOR_ISDN**, wenn weder eine X25-Adresse noch eine ISDN-Adresse angegeben wurde

=**RVSCAL_NEIGHBOR_STATION**, wenn bei "delete" Routing-Verbindungen zu dieser Station bestehen

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

### 11.2.2 Funktionen zur Verwaltung von rvs® Parameter

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von rvs® Parametern erforderlich sind.

#### 11.2.2.1 Typ-Definitionen

```
  typedef struct{
char  s_parm[RVSCAL_L_PARM_NAME];
  SINT  i_type ;
  long  len ;
  char  s_val[RVSCAL_L_PARM_VAL] ;
 } PARM_STRUCT ;
```

#### 11.2.2.2 Parameterwerte aus der Datenbank holen

**Prototyp `rvsGetParm`:**

```
PROCDEF int PROCKEYW rvsGetParm( char *parm, PARM_STRUCT
*stparm);
```

**Beschreibung der Parameter**

**parm** (char *, input)

Parametername

**stparm** (**PARM_STRUCT** *, output)

Struktur mit Informationen über den Parametereintrag

#### 11.2.2.3 Nächsten Parameter aus der Datenbank holen

**Prototyp `rvsGetNextParm`:**

```
PROCDEF int PROCKEYW rvsGetNextParm( char *parm, PARM_STRUCT
*stparm);
```

**Beschreibung der Parameter**

**parm** (char *, input)

Vorheriger Parameter

**stparm** (**PARM_STRUCT** *, output)

Struktur mit Informationen über den nächsten Parametereintrag

#### 11.2.2.4 Parameterwert in die Datenbank schreiben

**Prototyp `rvsWriteParm`:**

```
PROCDEF int PROCKEYW rvsWriteParm( char *parm, PARM_STRUCT
*stparm);
```

**Beschreibung der Parameter**

**parm** (char *, input)

Parametername

**stparm** (**PARM_STRUCT** *, input)

Struktur mit Informationen über den Parameter-eintrag

#### 11.2.2.5 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

### 11.2.3 Funktionen zur Verwaltung von rvs® Operator-Kommandos

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von rvs® Operator-Kommandos erforderlich sind.

#### 11.2.3.1 Operator-Kommando in die Datenbank schreiben

**Prototyp `rvsStoreOK`:**

```
PROCDEF int PROCKEYW rvsStoreOK( char *command);
```

**Beschreibung der Parameter**

**command** (char *, input)

Kommando-String

#### 11.2.3.2 rvs® Monitor aufwecken

**Prototyp `rvsWakeMonitor`:**

```
PROCDEF int PROCKEYW rvsWakeMonitor( void);
```

**Beschreibung der Parameter**

keine Parameter vorhanden

#### 11.2.3.3 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_RC_WAKE_FAILED**, wenn das Aufweck-Kommando fehlschlägt

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

### 11.2.4 Funktionen zur Verwaltung von residenten Empfangseinträgen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von residenten Empfangseinträgen erforderlich sind.

#### 11.2.4.1 Typ-Definitionen und Makros

```
typedef struct{
char  uid_local[RVSCAL_L_USID];
char  vdsn[RVSCAL_L_VDSN];
char  uid_sender[RVSCAL_L_USID];
char  sid_sender[RVSCAL_L_STATID];
char  dsn_local[RVSCAL_L_DSN];
char  s_replace[RVSCAL_L_C1];
char  s_disp[RVSCAL_L_C1];
SINT  flg_stamp;
char  s_printdef[RVSCAL_L_C1];
char  dsn_batchjob[RVSCAL_L_DSN];
char  uid_creator[RVSCAL_L_USID];
```

```
  char  accnt_rcv[RVSCAL_L_ACCT];
  char  comment[RVSCAL_L_RECMNT];
  char  dt_lastused[RVSCAL_L_DT];
 } INFO_RE ;


#define RE_UPDATE    1
#define RE_DELETE    2
#define RE_CREATE    3
```

#### 11.2.4.2 Nächste Kommandonummer des residenten Empfangseintrages aus der Datenbank holen

**Prototyp `rvsGetNextRE`:**

```
PROCDEF int PROCKEYW rvsGetNextRE( const long cn_pre, long
*lpcn_re);
```

**Beschreibung der Parameter**

**cn_pre** (const long, input)

Kommandonummer des vorherigen residenten Empfangseintrages

**lpcn_re** (long *, output)

Kommandonummer des nächsten in der **RE**-Tabelle gefundenen residenten Empfangseintrages

#### 11.2.4.3 Residenten Empfangseintrag aus der Datenbank holen

**Prototyp `rvsGetRE`:**

```
PROCDEF int PROCKEYW rvsGetRE( const long cn_re, INFO_RE
*reinfo);
```

**Beschreibung der Parameter**

**cn_re** (const long, input)

Kommandonummer des residenten Empfangseintrages

**reinfo** (**INFO_RE** *, output)

Struktur mit Informationen über den residenten Empfangseintrag

#### 11.2.4.4 Residente Empfangseinträge konfigurieren

**Prototyp `rvsResidentResceiveEntry`:**

```
 PROCDEF int PROCKEYW rvsResidentReceiveEntry( const int icmd,
INFO_RE *reinfo);
```

**Beschreibung der Parameter**

**icmd** (const int, input)

Kommando zur Angabe, was getan werden soll

**RE_UPDATE** – überarbeitet den residenten Empfangseintrag
**RE_DELETE** – löscht den residenten Empfangseintrag
**RE_CREATE** – erstellet einen residenten Empfangseintrag

**reinfo** (INFO_RE *, input)

Struktur mit Informationen über den residenten Empfangseintrag

### 11.2.4.5 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_END_FETCH**, wenn es keinen übereinstimmenden residenten Empfang-seintrag

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_DSN**, wenn ein ungültiger **DSNNEW** angegeben wurde

=**RVSCAL_INVALID_JOB**, wenn ein ungültiger **JOB** angegeben wurde

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_INVALID_SID**, wenn **SID_SENDER** nicht bekannt ist

=**RVSCAL_NOT_PRIVILEGED**, wenn der Benutzer nicht privilegiert ist, residente Empfangseinträge zu konfigurieren

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet oder geschlossen werden konnte

=**RVSCAL_RE_NOT_FOUND**, wenn ein residenter Empfangseintrag nicht gefunden werden konnte

=**RVSCAL_DUPLICATE_RE**, wenn bei **icmd**=**RE_CREATE** ein doppelter residenter Empfangseintrag gefunden wurde

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

### 11.2.5 Funktionen zur Verwaltung von Einträgen für Jobstart nach Sendeversuch

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Einträgen für Jobstart nach Sendeversuch erforderlich sind.

#### 11.2.5.1 Typ-Definitionen und Makros

```
  typedef struct{
  char  vdsn[RVSCAL_L_VDSN];
  char  uid_sender[RVSCAL_L_USID];
  char  sid_receiver[RVSCAL_L_STATID];
  long  cnt_sendatt;
  char  dsn_batchjob[RVSCAL_L_DSN];
  char  uid_creator[RVSCAL_L_USID];
  char  comment[RVSCAL_L_RECMNT];
  char  dt_lastused[RVSCAL_L_DT];
 } INFO_JS ;
#define JS_UPDATE    1
#define JS_DELETE    2
#define JS_CREATE    3
```

#### 11.2.5.2 Nächste Kommandonummer des Eintrages für Jobstart aus der Datenbank holen

**Prototyp `rvsGetNextJS`:**

```
PROCDEF int PROCKEYW rvsGetNextJS( const long cn_pre, long
*lpcn_js);
```

**Beschreibung der Parameter**

**cn_pre** (const long, input)

Kommandonummer des vorherigen Eintrages

**lpcn_js** (long *, output)

Kommandonummer des in der **JS**-Tabelle gefundenen Eintrages für Jobstart nach Sendeversuch

#### 11.2.5.3 Jobstarteintrag aus der Datenbank holen

**Prototyp `rvsGetJS`:**

```
PROCDEF int PROCKEYW rvsGetJS( const long cn_js, INFO_JS
*jsinfo);
```

**Beschreibung der Parameter**

**cn_js** (const long, input)

Kommandonummer des Jobstarteintrages

**jsinfo** (**INFO_JS** *, output)

Struktur mit Informationen über den Jobstarteintrag

### 11.2.5.4 Eintrag für Jobstart nach Sendeversuch konfigurieren

**Prototyp `rvsJobStartEntry`:**

```
PROCDEF int PROCKEYW rvsJobStartEntry( const int icmd, INFO_JS
*jsinfo);
```

**Beschreibung der Parameter**

**icmd** (const int, input)

Kommando zur Angabe, was getan werden soll

**JS_UPDATE** – überarbeitet einen Jobstarteintrag
**JS_DELETE** – löscht einen Jobstarteintrag
**JS_CREATE** - erstellt einen Jobstarteintrag

**jsinfo** (**INFO_JS** *, input)

Struktur mit Informationen über den Jobstarteintrag

#### 11.2.5.5 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_END_FETCH**, wenn es keinen übereinstimmenden Jobstarteintrag gibt

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_DSN**, wenn ein ungültiger **DSNNEW** angegeben wurde

=**RVSCAL_INVALID_JOB**, wenn ein ungültiger **JOB** angegeben wurde

=**RVSCAL_INVALID_NAME**, wenn der Parametername nicht vorhanden ist

=**RVSCAL_INVALID_SID**, wenn **SID_RECEIVER** nicht bekannt ist

=**RVSCAL_NOT_PRIVILEGED**, wenn der Benutzer nicht privilegiert ist, Einträge für Jobstart nach Sendeversuch zu konfigurieren

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet oder geschlossen werden konnte

=**RVSCAL_JS_NOT_FOUND**, wenn der Jobstarteintrag nicht gefunden werden konnte

=**RVSCAL_DUPLICATE_JS**, wenn bei **icmd** = **JS_CREATE** ein doppelter Jobstarteintrag gefunden wurde

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

### 11.2.6 Funktionen zur Verwaltung von Benutzereinträgen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von Benutzereinträgen erforderlich sind.

#### 11.2.6.1 Typ-Definitionen und Makros

```
typedef struct{
  char  uid[RVSCAL_L_USID];
  char  s_priv[RVSCAL_L_C1];
  char  s_prof[RVSCAL_L_C1];
  char  s_lang[RVSCAL_L_LANG];
  char  s_name[RVSCAL_L_USERNAME];
 } INFO_USER ;
```

```
#define USER_UPDATE    1
#define USER_DELETE    2
#define USER_CREATE    3
```

#### 11.2.6.2 Nächsten Benutzer aus der Datenbank holen

**Prototyp `rvsGetNextUser`:**

```
PROCDEF int PROCKEYW rvsGetNextUser( char *userpre, char
*user);
```

**Beschreibung der Parameter**

**userpre** (char *, input)

vorheriger Benutzername

**user** (char *, output)

Nächster Benutzername in der BT-Tabelle

#### 11.2.6.3 Benutzereintrag aus der Datenbank holen

**Prototyp `rvsGetUser`:**

```
PROCDEF int PROCKEYW rvsGetUser( char *user, INFO_USER
*usinfo);
```

**Beschreibung der Parameter**

**user** (char *, input)

Benutzername

**usinfo** (**INFO_USER** *, output)

Struktur mit Informationen über den Benutzereintrag

#### 11.2.6.4 Benutzereintrag konfigurieren

**Prototyp `rvsUser`:**

```
PROCDEF int PROCKEYW rvsUser( int icmd, INFO_USER *usinfo);
```

**Beschreibung der Parameter**

**icmd** (const int, input)

Kommando zur Angabe, was getan werden soll

**USER_UPDATE** – überarbeitet einen Benutzer
**USER_DELETE** – löscht einen Benutzer
**USER_CREATE** – erstellt einen Benutzer

**usinfo** (**INFO_USER** *, output)

Struktur mit Informationen über den Benutzereintrag

#### 11.2.6.5 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_END_FETCH**, wenn es keinen übereinstimmenden Benutzereintrag gibt

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_INVALID_OWN_PRIV**, wenn der Benutzer seine Privilegien senken will

=**RVSCAL_INVALID_UID**, wenn der Parameter **UID** leer ist

=**RVSCAL_INVALID_USER**, wenn der Benutzer nicht vorhanden ist oder sich selbst zu löschen versucht

=**RVSCAL_NOT_PRIVILEGED**, wenn der Benutzer nicht privilegiert ist, Benutzereinträge zu konfigurieren

=**RVSCAL_DBERROR**, wenn die Datenbank nicht geöffnet oder geschlossen werden konnte

=**RVSCAL_USER_NOT_FOUND**, wenn der Benutzereintrag nicht gefunden werden konnte

=**RVSCAL_DUPLICATE_USER**, wenn bei **icmd** = **USER_CREATE** ein doppelter Benutzereintrag gefunden wurde

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

### 11.2.7 rvs® Datenbank Funktionen

Dieses Kapitel beschreibt die Funktionen, die für die Verwaltung von rvs® Datenbankfunktionen erforderlich sind.

#### 11.2.7.1 Typ-Definitionen und Makros

```
#define RVSCAL_PIPE_NAME "\\\\.\\pipe\\rvsdb"
#define RVSCAL_OLEVENT_NAME "rvsdb_olevent"
#define RVSCAL_PIPE_TIMEOUT  90000
#define RVSCAL_L_OLEVENT    14
#define RVSCAL_L_PIPENAME   15
#define DEL_DB         0x01
```

```
#define DEL_LOG        0x02
#define DEL_TMP        0x04
#define DEL_REMDB       0x08
#define DUMP_RES        0x01
#define DUMP_USER       0x02
#define DUMP_JS        0x04
#define DUMP_STATION      0x08
#define DUMP_RU_ALL      DUMP_RES | DUMP_USER | DUMP_JS |
DUMP_STATION
```

#### 11.2.7.2 Datenbank speichern

**Prototyp `rvsDumpDB`:**

```
PROCDEF int PROCKEYW rvsDumpDB( char *environment, char *dsn);
```

**Beschreibung der Parameter**

**environment** ( char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung

**dsn** ( char *, input)

Dateiname der zu speichernden Datei

#### 11.2.7.3 Datenbank wiederherstellen

**Prototyp `rvsWriteDB`:**

```
PROCDEF int PROCKEYW rvsWriteDB( char *environment, char
*dsn);
```

**Beschreibung der Parameter**

**environment** ( char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung

**dsn** ( char *, input)

Dateiname der zu gespeicherten Datei

#### 11.2.7.4 Datenbank initialisieren

**Prototyp `rvsInitDB`:**

```
PROCDEF int PROCKEYW rvsInitDB( char *environment, char
*sid_loc);
```

**Beschreibung der Parameter**

**environment** ( char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung

**sid_loc** ( char *, input)

lokale Stations-ID

### 11.2.7.5 Datenbank löschen

**Prototyp `rvsDeleteDB`:**

```
PROCDEF int PROCKEYW rvsDeleteDB( char *environment, const int
delattrib);
```

**Beschreibung der Parameter**

**environment** ( char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer nullterminierter String ist, sucht rvs® nach der Standardumgebung

**delattrib** ( const int, input)

Das Attribut gibt an, welche Datei gelöscht werden soll. Mögliche Werte sind:

**DEL_DB** (Standard) – alle Datenbankdateien löschen,

**DEL_LOG** – Logdateien entfernen,

**DEL_TMP** – alle Dateien aus dem temporären rvs® Verzeichnis entfernen

### 11.2.7.6 Benutzer-, Empfangs-, Jobstart- und Stationseinträge speichern

**Prototyp `rvsDumpRU`:**

```
PROCDEF int PROCKEYW rvsDumpRU( char *environment, char *dsn,
const int dumpattrib);
```

**Beschreibung der Parameter**

**environment** ( char *, input)

Name der Umgebungsdatei

Wenn der Wert **NULL** oder ein leerer

nullterminierter String ist, sucht rvs® nach der Standardumgebung

**dsn** ( char *, input)

Dateiname der zu speichernden Datei. Die Dateiinhalte erhalten das für die rvs® Kommandozeilen-Schnittstelle einsetzbare Eingabeformat.

**dumpattrib** ( const int, input)

Das Attribut gibt an, welche Datei wiederhergestellt werden soll. Mögliche Werte sind:

**DUMP_USER** (Standard) – alle Benutzereinträge speichern,

**DUMP_RES** – alle residenten Empfangseinträge speichern
**DUMP_JS** – alle für Jobstart nach Sendeversuch speichern
**DUMP_STATION** – alle Stationseinträge speichern

#### 11.2.7.7 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_DSN_NOT_EXIST**, wenn die angegebene Datei nicht vorhanden ist

=**RVSCAL_INTERNAL_ERROR**, wenn ein interner Datenbankfehler aufgetreten ist

#### 11.2.7.8 Versionsnummer der rvs® Datenbank lesen

**Prototyp `rvsGetVersion`:**

```
PROCDEF int PROCKEYW rvsGetDBVersion( char *pszDBVersion);
```

**Beschreibung der Parameter**

**pszDBVersion** ( char *, output)

Version der rvs® Datenbank

#### 11.2.7.9 Rückgabewerte

**FUNCTIONVALUE** (int)

=**RVSCAL_OK**, wenn die Funktion erfolgreich ist

=**RVSCAL_PARAMETER_CHECK**, wenn mindestens ein Parameter falsch ist

=**RVSCAL_ENVIRONMENT_NOT_EXIST**, wenn die Umgebungsdatei nicht vorhanden ist

=**RVSCAL_ERROR_GETVERSION**, Version kann nicht bestimmt werden

### 11.2.8 Andere Funktionen

Dieses Kapitel beschreibt, wie Sie SID aus der ODETTE-ID erzeugen und umgekehrt und wie Sie den Listenstatus der rvs® Kommandos erhalten.

#### 11.2.8.1 SID aus der ODETTE ID erzeugen oder umgekehrt

**Prototyp `rvsgetsid`:**

```
char *rvsgetsid(char *s_odette_id);
```

**Beschreibung der Parameter**

**FUNCTIONVALUE** (char *)

=**NULL**, wenn kein **SID** gefunden wurde oder bei einem **DB** Fehler.

=Zeiger zum String, der rvs® **SID** enthält

**s_odette_id** (char *, input)

Zeiger zum String, der die ODETTE ID (max. 26 Bytes) **SID** enthält

**Prototyp `rvsgetodid`:**

```
char *rvsgetodid(char *s_sid);
```

**Beschreibung der Parameter**

**FUNCTIONVALUE** (char *)

=**NULL**, wenn keine ODETTE ID gefunden wurde oder bei einem **DB** Fehler

=Zeiger zu einem maximal 26 Bytes langen String

**s_sid** (char *, input)
Zeiger zum String, der rvs® **SID** enthält

### 11.2.8.2 Auflisten der Status der rvs® Kommandos

**Prototyp `rvslistcmd`:**

```
typedef struct {
  char status;
  short errorcode;
  } RVSCMD;
int rvslistcmd(long l_cmdid, RVSCMD *p_info);
```

**Beschreibung der Parameter**

| | |
|---|---|
| **FUNCTIONVALUE** | (int)<br>=**RVSCAL_OK**, wenn `rvslistcmd` erfolgreich ist |
| **l_cmdid** | (long, input)<br>Die **CMDID** (Kommandonummer) des Kommandos, welches bearbeitet wurde (Rückgabewert von `rvscal`). |
| **p_info** | (struct **RVSCMD** *, output)<br>Zeiger auf eine Struktur, welche Informationen über das Kommando **CMDID** enthält. |
| **RVSCMD.status** | (char, output)<br>Zeichen, das den Status des Kommandos enthält: **a**/**d**/**e**/**h**/**p**/**q**/**s**/**f**/**...** |
| **RVSCMD.errorcode** | (short, output)<br>Short Integer, welche einen Fehler-Code enthält, wenn dieser von **0** abweicht**.** |

# 12 Glossar

### ASCII

American National Standard Code for Information Interchange

### Access Method

Die Access Method (Zugangsmethode) beschreibt die Art und Weise, wie zwei Stationen miteinander verbunden sind.

### Dialog-Schnittstelle (`rvsdia`)

Die Dialog-Schnittstelle von rvs® stellt interaktive Benutzerfunktionen zur Verfügung.

### EBCDIC

Extended Binary Coded Decimal Interchange Code

### EDI

**E**lectronic **D**ata **I**nterchange

### EDIFACT

**E**lectronic **D**ata **I**nterchange for **A**dministration, **C**ommerce and **T**ransport

### EERP

End-to-End-Response. ODETTE Ausdruck für die Quittung am Ende der Übertragung bei der Sendeübertragung.

### ET

**E**mpfänger**T**abelle

Interne rvs® Tabelle, die die Empfänger beschreibt.

### ETSI

European Telecommunications Standardization Institute

**HPFS**

**H**igh **P**erformance **F**ile **S**ystem (OS/2)

Datei-System von OS/2.

**IE**

**I**nformations**E**ingang

Internes rvs® Kommando zur Steuerung der Zustellung und des Routing von empfangenen Dateien.

**IZ**

**I**nformations**Z**ustellung

Internes rvs® Kommando für die Zustellung empfangener Dateien an einen lokalen Empfänger.

**Kommandozeilen-Schnittstelle (`rvsbat`)**

Die Kommandozeilen-Schnittstelle von rvs® stellt Funktionen für den Ablauf von Prozessen im Hintergrund zur Verfügung.

**Kommunikations-Modul (`rvscom`)**

Das Kommunikations-Modul von rvs® verbindet verschiedene Stationen miteinander und versendet und empfängt Dateien.

**MasterTransmitter (`rvsxmt`)**

Der MasterTransmitter des rvs® koordiniert Sende- und Empfangsprozesse, um die optimale Auslastung der Netzkapazität zu gewährleisten.

**Monitor (`rvsmon`)**

Der rvs® Monitor ist die Hauptkomponente des rvs® Systems. Er kontrolliert Sende- und Empfangsübertragungen und startet automatisch Jobs (RE, JS), wenn nötig,

**ODETTE**

Organization for Data Exchange by Tele Transmission in Europe

Die komplette Beschreibung von OFTP erhalten Sie unter:

http://www.odette.org/

**OFTP**

ODETTE File Transfer Protokoll

Das ODETTE File Transfer Protokoll ist die Definition des File Transfer Protokolls von der ODETTE Group IV für die ISO/OSI Schichten 4 bis 7.

Internationales Protokoll, das in vielen Geschäftsbereichen benutzt wird (z.B. Industrie, Commerce, Finanzen).

### Operator-Konsole (`rvscns`)

Die Operator-Konsole stellt dem Administrator rvs® Funktionen zum Kontrollieren des rvs® Systems zur Verfügung.

### OSI

Open System Interconnection

### PDF

Portable Document Format (Portables Dokumentenformat)

### Protokoll

Wenn zwei verschiedene Computer miteinander verbunden werden, müssen sie mit dem gleichen Protokoll arbeiten. Dieses Protokoll definiert Aktionen und Reaktionen als auch die "Sprache", mit der sich die beiden Computer miteinander verständigen.

### RE

**R**esidenter **E**mpfangseintrag

rvs® Tabelle für die Beschreibung von Aktionen, die nach dem Eintreffen einer einzelnen Datei für einen lokalen Empfänger starten soll.

### `rvsmon`

Siehe Monitor

### SE

**S**ende**E**intrag

Internes rvs® Kommando zur Steuerung des Dateiversands.

### SID

rvs® Ausdruck für die Stations-ID

### SK

**S**ende**K**ommando

Internes rvs® Kommando zur Steuerung der Übertragung einer Datei an einen Nachbarknoten.

### Station

Eine Station ist ein Knoten in einem rvs® Netzwerk. Jede Station in Ihrem rvs® System ist durch eine eindeutige Stations-ID (SID) identifiziert.

### Übertragungskomponente

Kontrollprogramm und Leitungstreiber für eine spezielle Zugangsmethode

### VDA

Verband der Deutschen Automobilhersteller

Adresse:
Verband der Automobilindustrie e.V. (VDA)
Abt. Logistik
Postfach 17 05 63
60079 Frankfurt
Tel.: 069-7570-0

### VDSN

**V**irtual **D**ata **S**et **N**ame

ODETTE Ausdruck für den Namen einer Datei, die mit OFTP übertragen wurde.

# 13 Index